US006625624B1

(12) **United States Patent**
Chen et al.

(10) **Patent No.: US 6,625,624 B1**
(45) **Date of Patent: Sep. 23, 2003**

(54) **INFORMATION ACCESS SYSTEM AND METHOD FOR ARCHIVING WEB PAGES**

(75) Inventors: **Yih-Farn Robin Chen**, Bridgewater, NJ (US); **Chung-Hwa Herman Rao**, Metuchen, NJ (US); **Ming-Feng Chen**, Hsinchu (TW)

(73) Assignee: **AT&T Corp.**, New York, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/475,556**

(22) Filed: **Dec. 30, 1999**

Related U.S. Application Data

(60) Provisional application No. 60/133,345, filed on May 10, 1999, provisional application No. 60/126,705, filed on Mar. 29, 1999, provisional application No. 60/118,651, filed on Feb. 4, 1999, and provisional application No. 60/118,367, filed on Feb. 3, 1999.

(51) **Int. Cl.**[7] **G06F 17/30**

(52) **U.S. Cl.** **707/204**; 707/10; 707/104.1; 707/203; 709/245

(58) **Field of Search** 707/4, 10, 100, 707/203, 204, 1, 104.1; 709/219, 245

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,806,062	A *	9/1998	Chen et al.	707/4
5,860,071	A *	1/1999	Ball et al.	707/100
5,898,836	A *	4/1999	Freivald et al.	707/513
5,978,842	A *	11/1999	Noble et al.	709/218
6,026,437	A *	2/2000	Muschett et al.	709/219
6,167,453	A *	12/2000	Becker et al.	709/245
6,199,071	B1 *	3/2001	Nielsen	707/204
6,249,795	B1 *	6/2001	Douglis	707/511
6,282,548	B1 *	8/2001	Burner et al.	707/104.1
6,289,346	B1 *	9/2001	Milewski et al.	707/10
2002/0016789	A1 *	2/2002	Ong	707/10
2002/0065800	A1 *	5/2002	Morlitz	707/1
2002/0073058	A1 *	6/2002	Kremer et al.	707/1

OTHER PUBLICATIONS

Douglis, Fred, "Experiences with the AT&T Internet Difference Engine," 22nd Int'l Conf. for the Resource Mgt and Performance Eval. of Enterprise Continuing Sys. (CMG96), 12–96, p. 1–14.*

Chen et al., "TopBlend: An Efficient Implementation of HtmlDiff in JAVA," WebNet'00, pp. 1–6.*

Chen et al., "CIAO: A Graphical Navigator for Software and Document Repositories," IEEE, 1995, pp. 66–75.*

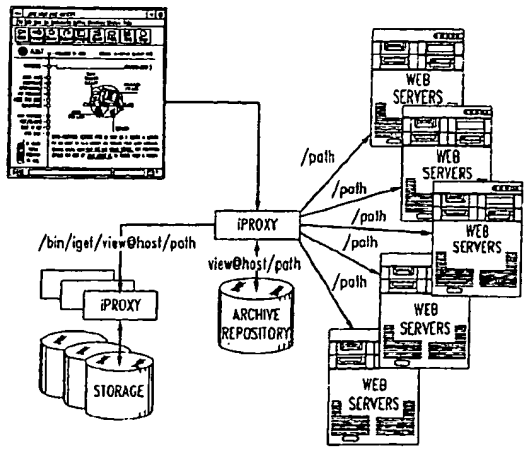
* cited by examiner

Primary Examiner—Charles Rones

(57) **ABSTRACT**

The present invention presents a system and method of providing information retrieved from a server from across a communication network that enables archiving services that do not interfere with existing components and protocols. The services enable users to retrieve and/or search for old information, even after such information has evolved or disappeared from the original server. The network resource naming (e.g. URL) format is extended to include archive directives that are intercepted and performed by a proxy server. The proxy interprets the archive directive and executes the specified archival command: e.g., adding the information to a storage repository, searching and retrieving the information from the storage repository, scheduling automatic archiving of specified server information, transparent archiving of information that is accessed by the client or of the client's cache. The information can be easily indexed by a timestamp. Multiple proxy servers can collaborate permitting information to be archived in a distributed fashion. Embodiments of the present invention advantageously do not require changes to client or server software or communication protocols.

10 Claims, 16 Drawing Sheets



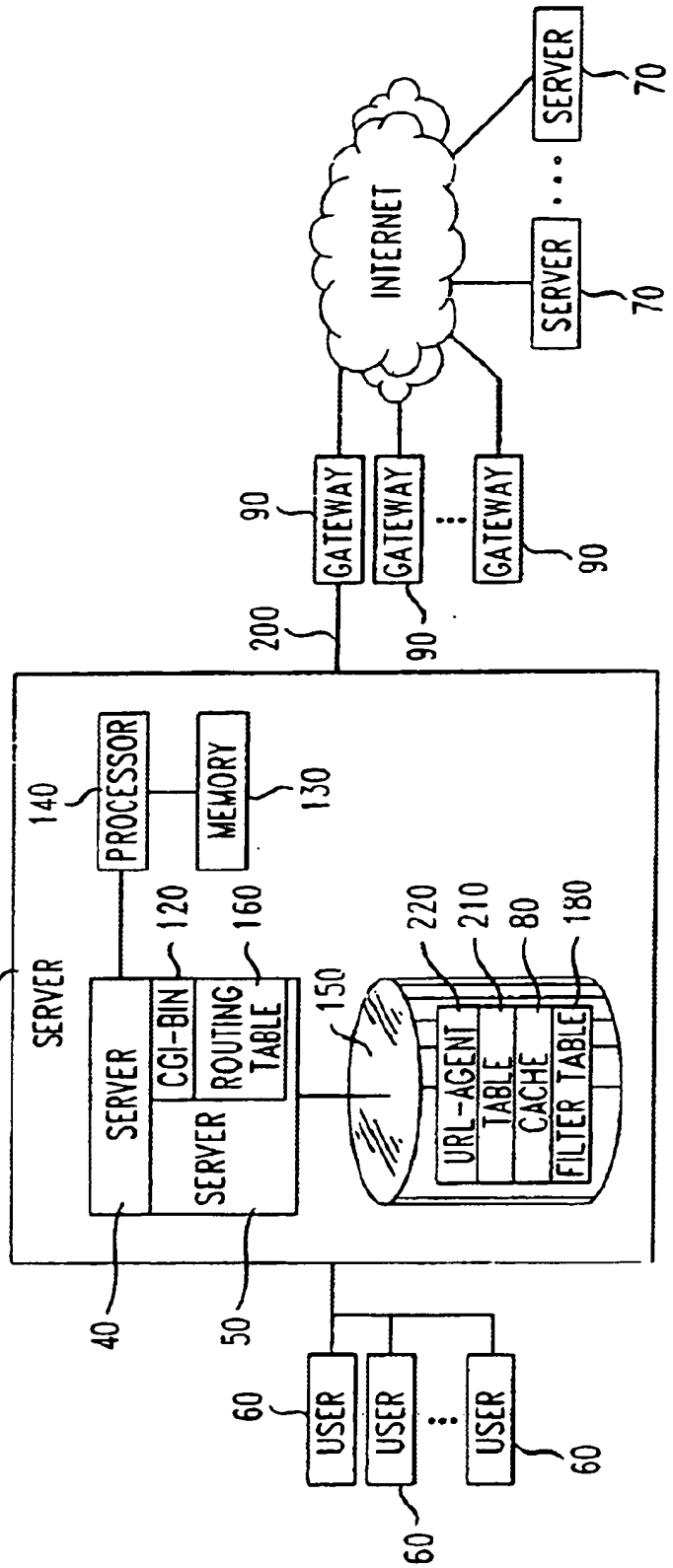

SERVER
70
INTERNET
SERVER
70
GATEWAY
90
GATEWAY
90
GATEWAY
90
200
90
30
SERVER
140
PROCESSOR
MEMORY
130
120
160
150
220
210
80
180
SERVER
CGI-BIN
ROUTING TABLE
SERVER
URL-AGENT
TABLE
CACHE
FILTER TABLE
40
50
60
USER
USER
USER
60
60

FIG. 1

FIG. 2

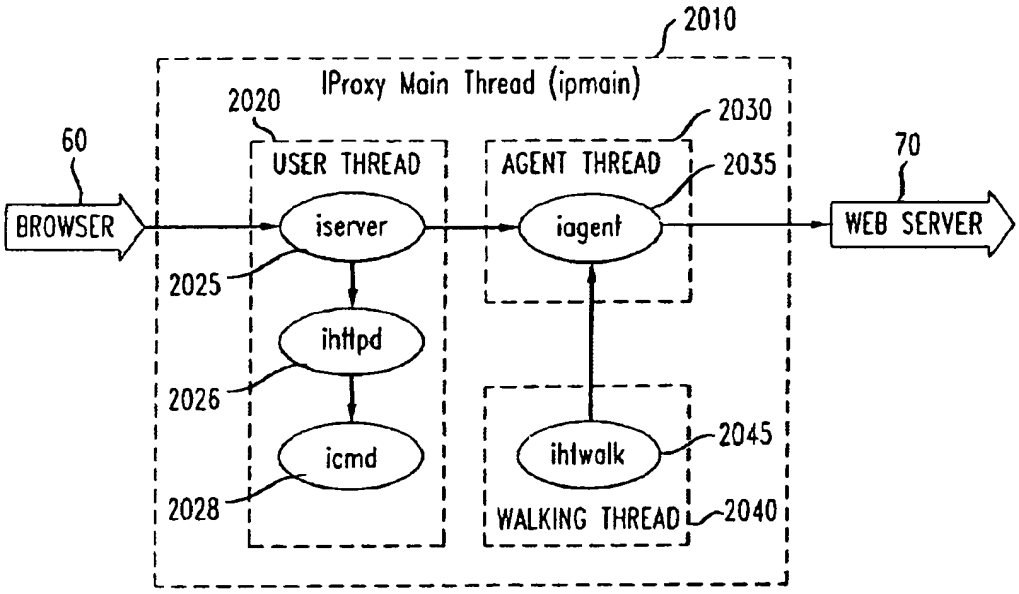

2010
IProxy Main Thread (ipmain)
2020
2030
60
USER THREAD
AGENT THREAD
2035
70
BROWSER
iserver
iagent
WEB SERVER
2025
ihttpd
2026
icmd
2028
ihtwalk
2045
WALKING THREAD
2040

FIG. 3

System Command	Description
reset & resetall	Reset the system. The system will reload the external configuration file and reset to the initial state.
connect=[On[Off]	Run the system as on-line mode (On) or off-line mode (Off). For the off-line mode, the system won't create "Agent Thread" for retrieving web pages. Instead, it responds to the requests with local cache files and logs those requests that cannot find pages in the cache.
setnet=netid	Switch the network to a new one "netid". This feature is used when the user is using more than one dial-in network for Internet access. If netid does not match the id in the control file, the system will call a CGI -bin to dial-in a different network.
dns=dns_server	Create a DNS agent on the local host. It then forwards local DNS requests to dns server.
forword=loaclport,rhost:rport[,proxyhost:port]	Create a TCP forwarding service. Connection to localport will be mapped to rhost:rport, and traffic to the former will be forwarded to the latter. This command uses "SSL proxy" protocol to connect to proxyhost:port.
pasv=rhost:rport	Create a passive channel to another server (running on rhost:rport).
htwalk= level[, -option]*[function[=args]]*	**level:** specify the depth of hyper reference hierarchy the system will walk down. **option:** options include: (a). Retrieving image files *(image)* or not *(noimage)*; (b). Accessing only local pages *(loacl)* or any Web pages *(global)*. **function:** the javabin function that is called for each visited page.

FIG. 4

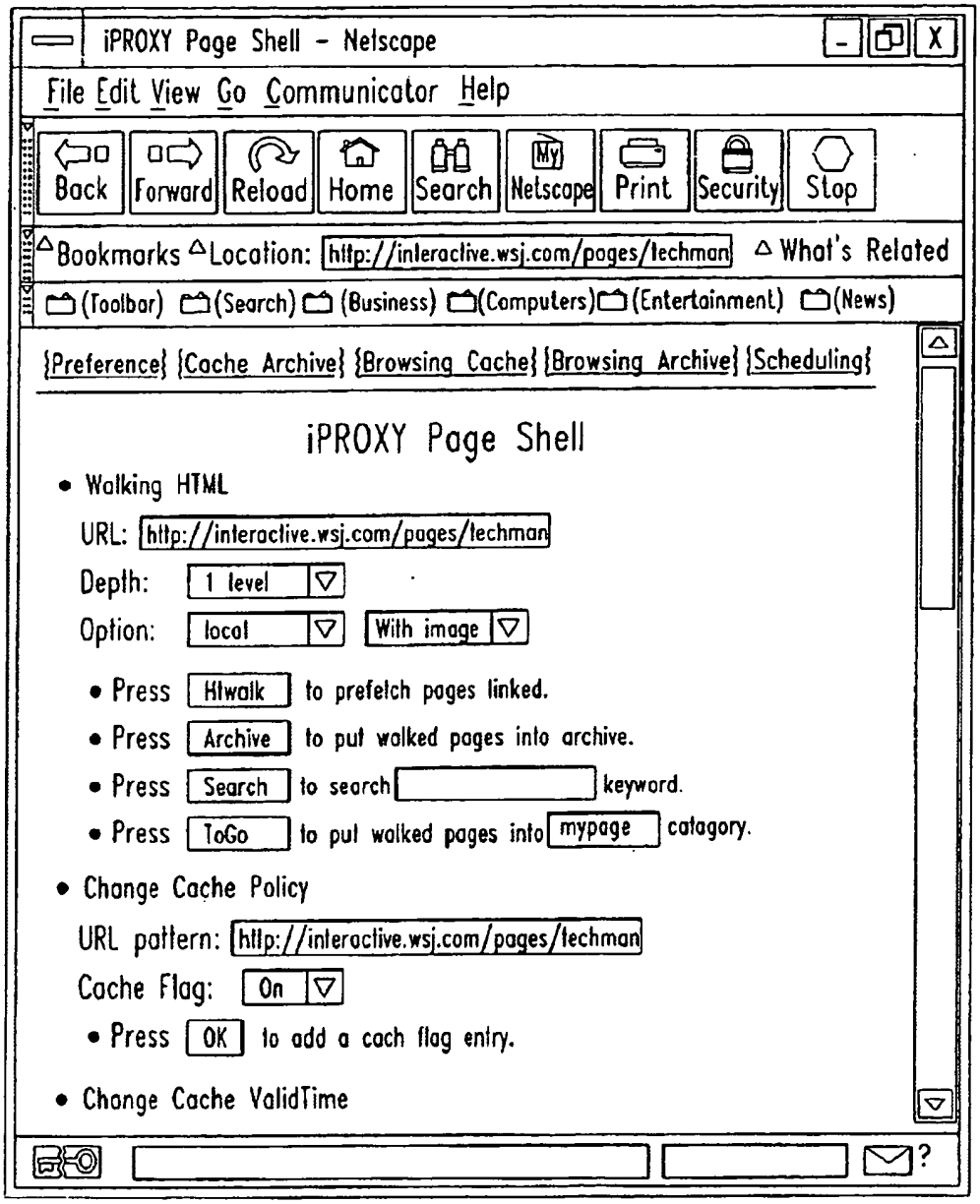

iPROXY Page Shell - Netscape
File Edit View Go Communicator Help
Back
Forward
Reload
Home
Search
Netscape
Print
Security
Stop
Bookmarks
Location: http://interactive.wsj.com/pages/techman
What's Related
(Toolbar)
(Search)
(Business)
(Computers)
(Entertainment)
(News)
{Preference} {Cache Archive} {Browsing Cache} {Browsing Archive} {Scheduling}
iPROXY Page Shell
Walking HTML
URL: http://interactive.wsj.com/pages/techman
Depth: 1 level
Option: local
With image
Press Htwalk to prefetch pages linked.
Press Archive to put walked pages into archive.
Press Search to search keyword.
Press ToGo to put walked pages into mypage catagory.
Change Cache Policy
URL pattern: http://interactive.wsj.com/pages/techman
Cache Flag: On
Press OK to add a cach flag entry.
Change Cache ValidTime

FIG. 5

HTTP Extension	Description
CONNECT *rhost:rport HTTP/1.0*	SSL socket proxy command. The command establishes a TCP connection between the sender and receiver so that the traffic to the sender of the message will be forwarded via this connection to the receiver proxy, which then forwards them to *rhost:rport.*
PASV *ServerProxy->connect host:port*	Receive a *passive* channel from ServerProxy. The passive channel will be used by Receiver (who is the ClientProxy in this case) to request Web pages. *"connect_host:port" is* the host and port of ClientProxy.
NEWPASV *connect host:port*	Request a new passive channel to *connect host:port.* Receiver will issue PASV command to *connect_host:port.*
MULTI *separator ... separator*	Reuse the connection.

02/02/2004, EAST Version: 1.4.1

FIG. 6

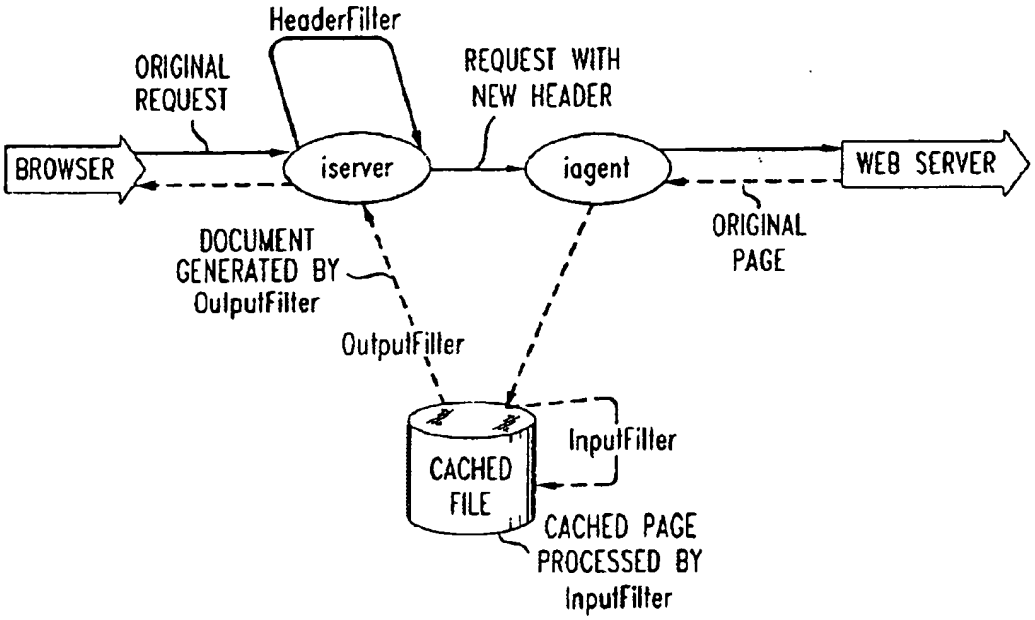

HeaderFilter
ORIGINAL REQUEST
REQUEST WITH NEW HEADER
BROWSER
iserver
iagent
WEB SERVER
ORIGINAL PAGE
DOCUMENT GENERATED BY OutputFilter
OutputFilter
InputFilter
CACHED FILE
CACHED PAGE PROCESSED BY InputFilter

FIG. 7A

```
public class inputFilterSample extends javabin {
   public Object Execute(OutputStream os,Hashtable ht)throws IOException {
                String url = (String)ht.get("url");
                String fname = (String)ht.get("path");
                byte orig_date[] = readCacheFile(fname);
                byte result_data[] = doInputFilter(orig_data);
                writeCacheFile(fname, result_data);
   }
   ...
}
```

FIG. 7B

```
public class outputFilterSample extends javabin {
   public Object Execute(OutputStream os,Hashtable ht)throws IOException {
                String url = (String)ht.get("url");
                String fname = (String)ht.get("path");
                byte cached_date[] = readCacheFile(fname);
                byte result_data[] = doOutputFilter(orig_data);
                os.write(result_data);
   }
   ...
}
```

FIG. 7C

```
public class headerFilterSample extends javabin {
   public Object Execute(OutputStream os,Hashtable ht)throws IOException {
                String orig_header = (String)ht.get("ht.header");
                String result_header = doHeaderFilter(orig_header);
                os.write(result_header.getBytes());
   }
   ...
}
```

FIG. 8

Statement	Description
	Null statement, used as comment.
":if exp" ..":elseif exp" ... ":else" ":endif"	Conditional statement. The format of "exp" is " str1 =str2" or "strf1=str2".
":reset"	Reload configuration and restart the proxy services.
":htwalk url argument"	Execute an HTML set walking immediately.
":forward lport,rhost:rport"	Create a TCP forwarding channel.
":pasv host:port"	Connect another proxy server and build a passive channel.
":block class_name arglist"..":endblock"	Invoke javabin.class name (with statements.)
":javabin class name arglist"	Invoke a javabin.class name.
":zip file filename"& ":unzip dump filename"	Compress/decompress an file.
:ctrlmacth var ctrl -type" ..":endctrl"	Match the entries with ctrl_type in ctrlfile.txt and store in ${var.id}, ${var.type), $(var.arg), and ${var.url) variables.
:ctrladd ctrl_type ctrl arg ctrl_url"	Add a new entry of ctrlfile.txt
:ctrldelete selected	Delete the entries whose ctrl_id is assigned in arguments.
:cronmatch var"... ":endcron"	Match the entries of scheduling and store in ${var.id), ${var.hour), ${var.wday), ${var.cmd), ${var.url), and ${var.arg) variables.
:cronadd hour wday cmd url walk level arglist	Add a new entry of scheduling.
":crondelete selected"	Delete the entries whose cron_id is assigned in arguments.
:sysinfo var (iforward liagent|ihtwalk|threads)" ... :endsys"	List the system information.

FIG. 9

```
Public class scriptFunctionSample extends javab in {
    public Object Execute(OutputStream os. Hashtable ht) throwb IO Exception {
        ic m d pC m d = (ic m d) ht.get ( ".pscript");
        Hashtable htVar = ( Hashtable) ht.get (".h tvar");
        StringTokenizer st = (StringTokenizer) ht.get (".argv");
        byte block[] = (byte [) ht.get (".block "); // = null if cmd is : Javabin
        String cmd = st.nextToken();
        if(cmd.equals ("xxx")) {
            htVar.put("foo ", value);
            pC md.callScript(block); // apply new value to script
        }
    }
}
```

FIG. 10

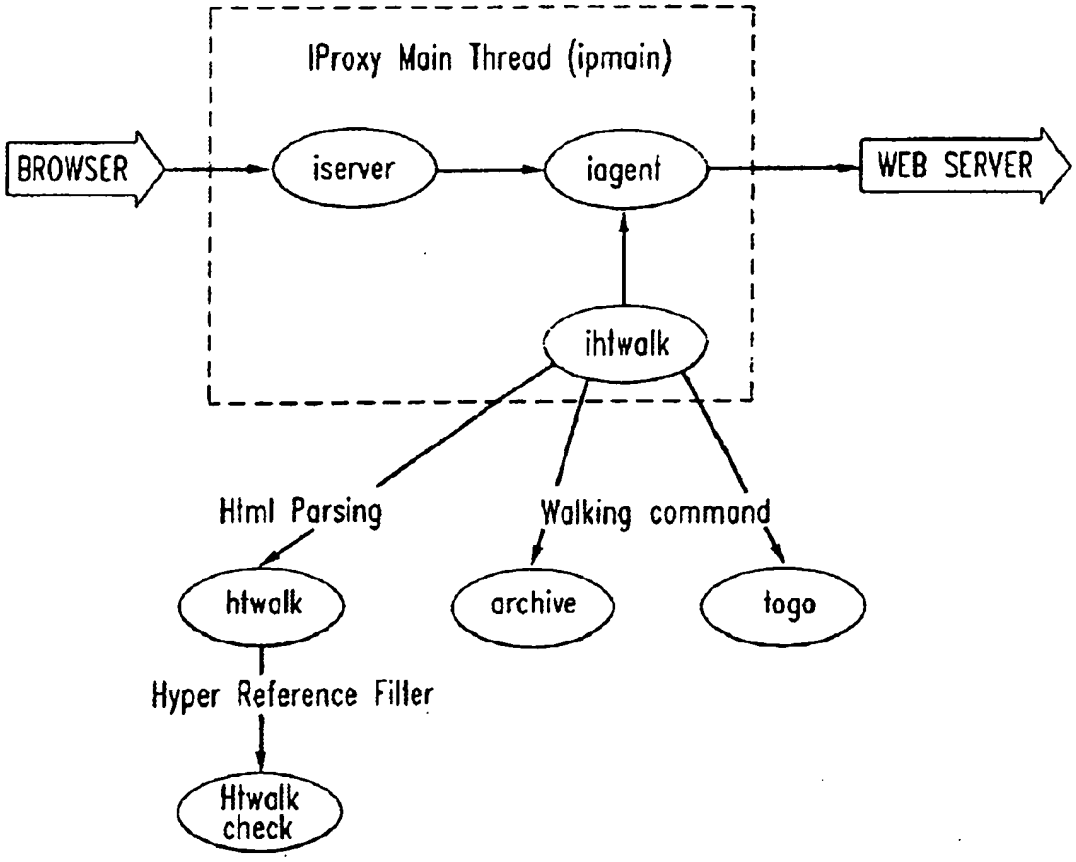

IProxy Main Thread (ipmain)
BROWSER
iserver
iagent
WEB SERVER
ihtwalk
Html Parsing
Walking command
htwalk
archive
togo
Hyper Reference Filter
Htwalk check

FIG. 11

```
public class htwalkGrepSample exents javabin {
  public void initProc() {
          // init process
  }
  public void finalProc() {
          // final process
  }
  public Object Execute (OutputStream os,Hashtable ht)throws IOException{
          initProc(); // init process
          String htcmd = (String)ht.get("htcmd");
          String url = (String)ht.get("url");
          String args = (Strin g)ht.get("args");
          String fname = (String)ht.get( "path");
          String key = (String)ht.get("walkopt");
          finalProc(); // final process
          if (keywordMatched(fnarne, key)) {
  ShowGrepResult(url, fname);
  }
          }
      i K
}
```

FIG. 12

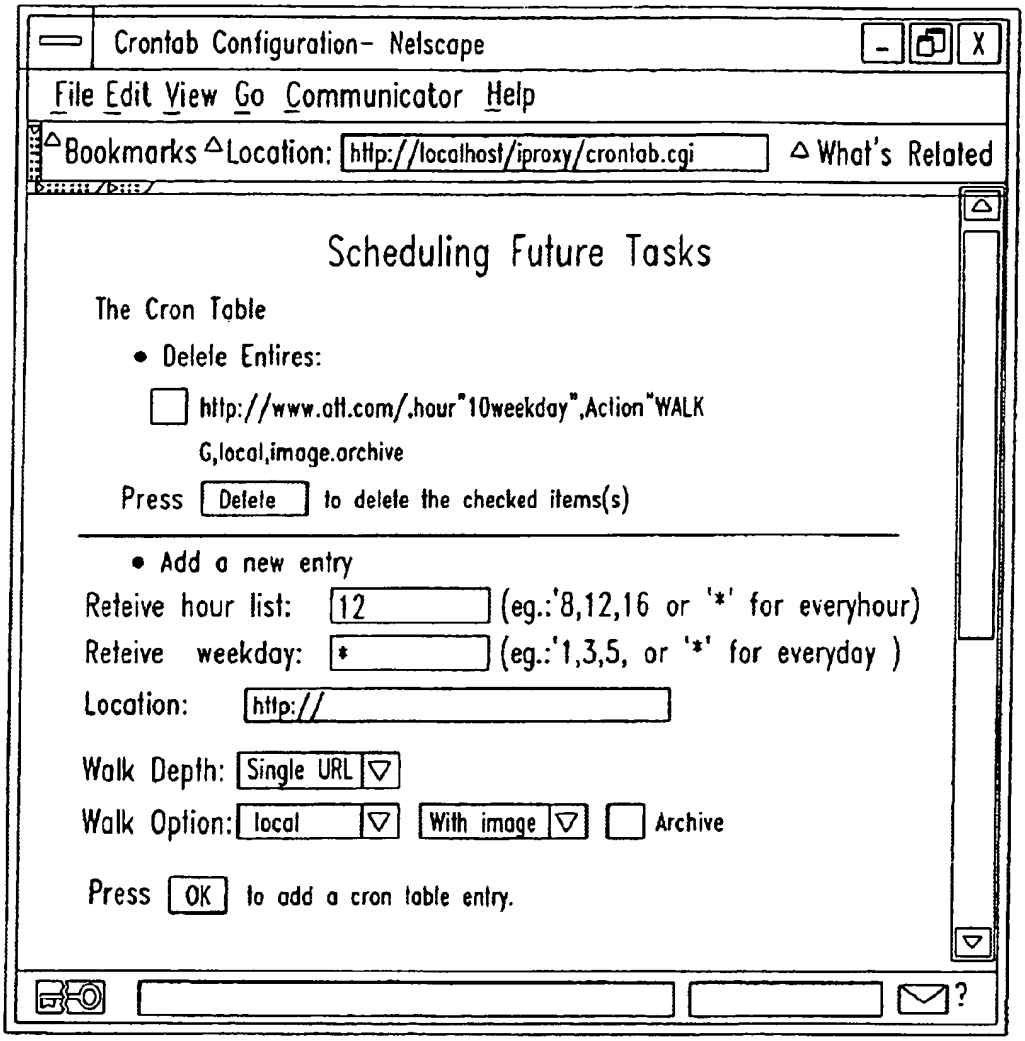

Crontab Configuration- Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://localhost/iproxy/crontab.cgi
What's Related
Scheduling Future Tasks
The Cron Table
• Delete Entires:
http://www.att.com/,hour"10weekday",Action"WALK
G,local,image.orchive
Press Delete to delete the checked items(s)
• Add a new entry
Reteive hour list: 12 (eg.:'8,12,16 or '*' for everyhour)
Reteive weekday: * (eg.:'1,3,5, or '*' for everyday)
Location: http://
Walk Depth: Single URL
Walk Option: local With image Archive
Press OK to add a cron table entry.

FIG. 13

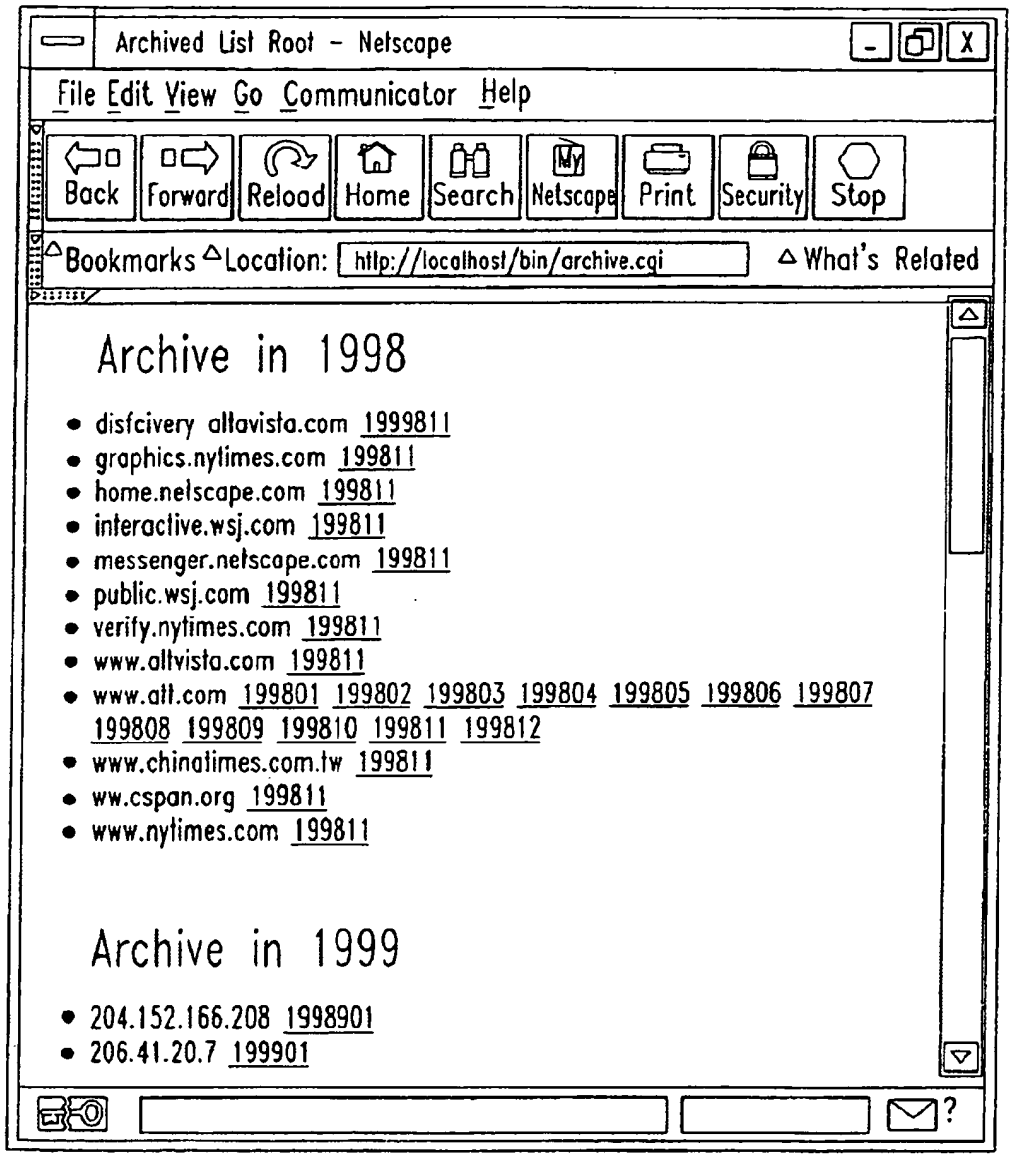

Archived List Root - Netscape
File Edit View Go Communicator Help
Back
Forward
Reload
Home
Search
Netscape
Print
Security
Stop
Bookmarks
Location: http://localhost/bin/archive.cgi
What's Related
Archive in 1998
disfcivery altavista.com 1999811
graphics.nytimes.com 199811
home.netscape.com 199811
interactive.wsj.com 199811
messenger.netscape.com 199811
public.wsj.com 199811
verify.nytimes.com 199811
www.altvista.com 199811
www.att.com 199801 199802 199803 199804 199805 199806 199807 199808 199809 199810 199811 199812
www.chinatimes.com.tw 199811
ww.cspan.org 199811
www.nytimes.com 199811
Archive in 1999
204.152.166.208 1998901
206.41.20.7 199901

FIG. 14

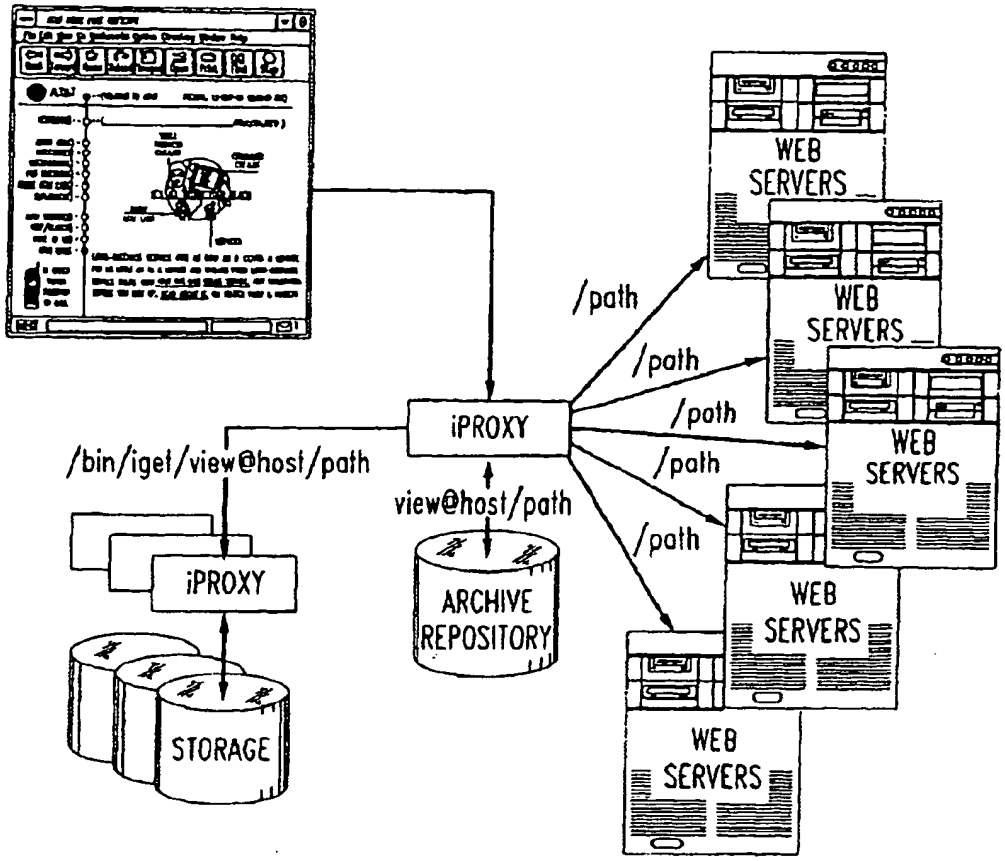


FIG. 15

```
<!-----------------with iproxy only ---------------->
<!-----:portal nop
          :portal version
          :portal to-read
          :portal dolog
          :portal top10
        :portal nop     --->
```

FIG. 16

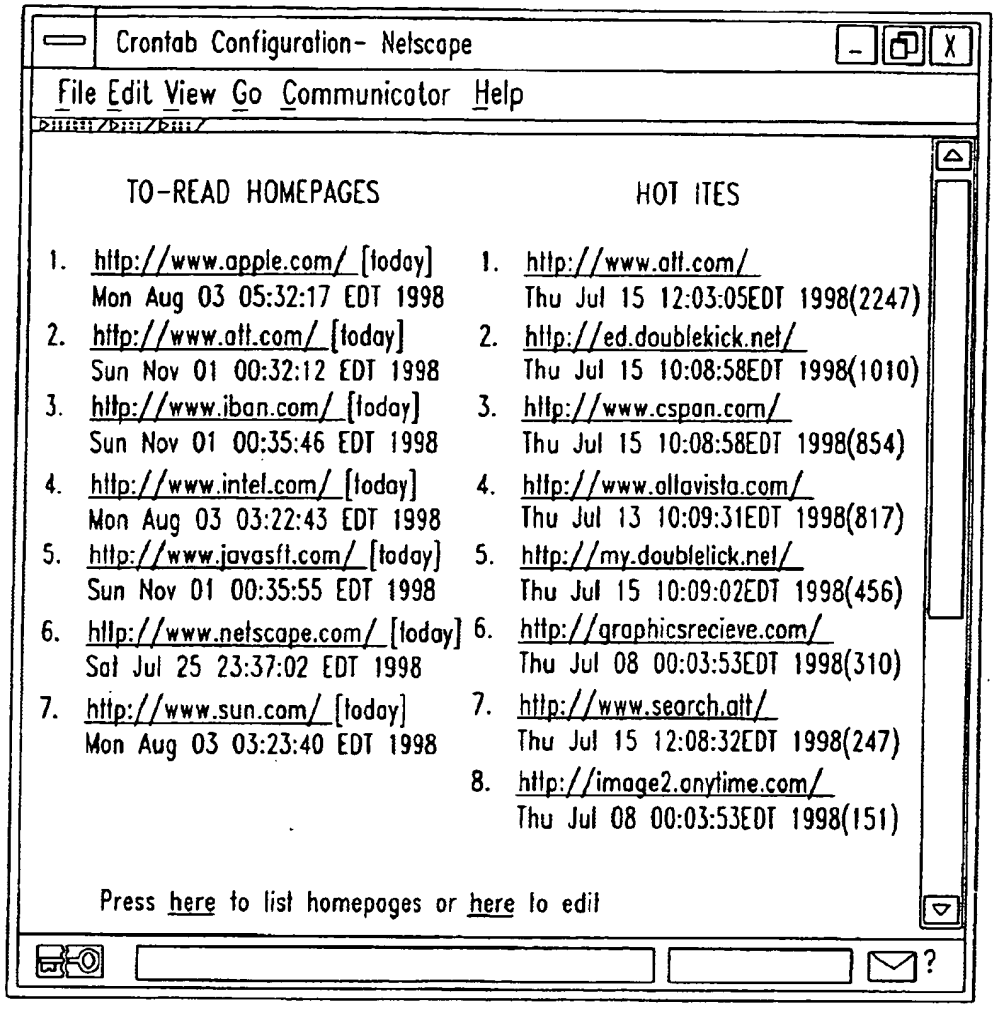

Crontab Configuration- Netscape
File Edit View Go Communicator Help
TO-READ HOMEPAGES
1. http://www.apple.com/ [today]
Mon Aug 03 05:32:17 EDT 1998
2. http://www.att.com/ [today]
Sun Nov 01 00:32:12 EDT 1998
3. http://www.iban.com/ [today]
Sun Nov 01 00:35:46 EDT 1998
4. http://www.intel.com/ [today]
Mon Aug 03 03:22:43 EDT 1998
5. http://www.javasft.com/ [today]
Sun Nov 01 00:35:55 EDT 1998
6. http://www.netscape.com/ [today]
Sat Jul 25 23:37:02 EDT 1998
7. http://www.sun.com/ [today]
Mon Aug 03 03:23:40 EDT 1998
HOT ITES
1. http://www.att.com/
Thu Jul 15 12:03:05EDT 1998(2247)
2. http://ed.doublekick.net/
Thu Jul 15 10:08:58EDT 1998(1010)
3. http://www.cspan.com/
Thu Jul 15 10:08:58EDT 1998(854)
4. http://www.altavista.com/
Thu Jul 13 10:09:31EDT 1998(817)
5. http://my.doublelick.net/
Thu Jul 15 10:09:02EDT 1998(456)
6. http://graphicsrecieve.com/
Thu Jul 08 00:03:53EDT 1998(310)
7. http://www.search.att/
Thu Jul 15 12:08:32EDT 1998(247)
8. http://image2.anytime.com/
Thu Jul 08 00:03:53EDT 1998(151)
Press here to list homepoges or here to edit

FIG. 17

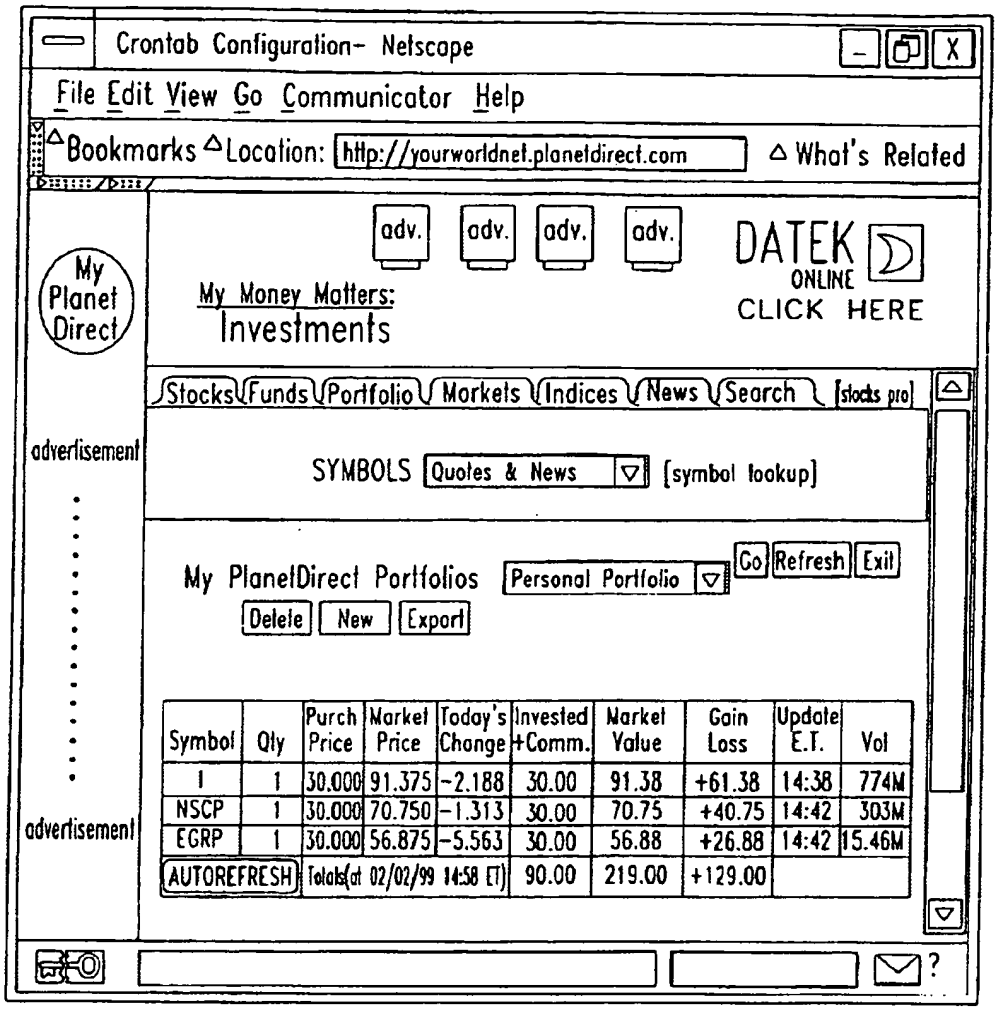


FIG. 18

Symbol	Qty	Purch Price	Market Price	Today's Change	Invested +Comm.	Market Value	Gain Loss	Update E.T.	Vol
I	257.0	36.5	91.375	-2.188	9745.5	24397.13	14651.63	14:38	774M
NSCP	50.0	21.0	70.750	-1.313	1069.95	3537.5	2467.55	14:42	303M
EGRP	40.0	39.38	56.875	-5.563	1595.15	2275.0	679.85	14:42	15.46M
AUTOREFRESH		Totals(at 02/02/99 14:58 ET)			12410.6	30209.63	17799.02		

INFORMATION ACCESS SYSTEM AND METHOD FOR ARCHIVING WEB PAGES

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims priority to Provisional Application Serial No. 60/118,367, "A Proxy-Based Personal Portal," filed on Feb. 3, 1999, and to Provisional Application Serial No. 60/118,651, "A Proxy-Based Personal Portal," filed on Feb. 4, 1999, and to Provisional Application Serial No. 60/126,705, "Archiving Web Pages with iProxy," filed on Mar. 29, 1999, and to Provisional Application Serial No. 60/133,345, "Proxy-Based Services," filed on May 10, 1999, the contents of which are incorporated by reference herein.

This application is also related to the Utility Patent Application, "Information Access System And Method," Ser. No. 08/994,600, filed on Dec. 19, 1997, now abandoned, the contents of which are incorporated by reference herein.

FIELD OF THE INVENTION

The present invention relates generally to information access systems, and, more particularly, to information access systems used to retrieve information from across a communication network.

BACKGROUND OF THE INVENTION

As more resources become available on communication networks such as the Internet, it has become increasingly more difficult to locate, manage, and integrate them. Many information retrieval clients such as the browsers by Netscape, Microsoft, and Mosaic have been introduced to make searching and retrieving information on the network more convenient and productive for end-users. Unfortunately, the closed architecture of such browsers has rendered the software design overloaded and monopolistic. Customizing the information retrieval process is difficult if not impossible.

For example, it would be advantageous to enable users to retrieve and search through old information—even after the original information has evolved or disappeared from the original server. Many websites provide timely content such as news that changes on a regular basis. A user may wish to revisit a web page that the user saw many months earlier, or search for information the user recalls seeing on a web page that the user visited in the past. Client browsers typically only provide a rudimentary save feature and lack basic archiving functionality. Any such functionality should be provided by some specialized application which must be installed or integrated with each client. Otherwise, the user must either have the foresight to explicitly save pages he or she might be interested in—or must rely on the content administrator to provide some archive and search facility.

SUMMARY OF THE INVENTION

The present invention presents a system and method of providing information retrieved from a server from across a communication network that enables archiving services that do not interfere with existing components and protocols. The services enable users to retrieve and/or search for old information, even after such information has evolved or disappeared from the original server. The network resource naming (e.g. URL) format is extended to include archive directives that are intercepted and performed by a proxy server. The proxy interprets the archive directive and executes the specified archival command: e.g., adding the information to a storage repository, searching and retrieving the information from the storage repository, scheduling automatic archiving of specified server information, transparent archiving of information that is accessed by the client or of the client's cache. The information can be easily indexed by a timestamp. Multiple proxy servers can collaborate permitting information to be archived in a distributed fashion. Embodiments of the present invention advantageously do not require changes to client or server software or communication protocols.

These and other advantages of the invention will be apparent to those of ordinary skill in the art by reference to the following detailed description and the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** sets forth a diagram of an information access system and method illustrating an embodiment of the present invention.

FIG. **2** sets forth an example of a system architecture illustrating an embodiment of the present invention.

FIG. **3** sets forth a list of system commands that can be invoked using URL extensions.

FIG. **4** sets forth a screenshot of an example service menu.

FIG. **5** sets forth a list of commands that can be invoked using HTTP extensions.

FIG. **6** sets forth a diagram illustrating the use of filters with an information access system and method.

FIG. **7A**, **7B** and **7C** set forth examples of cgi-bin filter programs.

FIG. **8** sets forth a list of scripting commands that can be used with a document pre-processing system.

FIG. **9** sets forth arguments for use with a javabin class invoked by the "block" and "javabin" commands in FIG. **8**.

FIG. **10** sets forth a diagram illustrating the use of archiving with an information system and method.

FIG. **11** sets forth an example of a grep class for use with a walking facility.

FIG. **12** sets forth a screenshot of an example cron menu.

FIG. **13** sets forth a screenshot of an example archiving menu.

FIG. **14** sets forth a diagram illustrating the use of multiple archiving repositories.

FIG. **15** sets forth an example of scripting directives embedded in an HTML comment.

FIG. **16** sets forth a screenshot of an example personalized portal.

FIG. **17** sets forth a screenshot of an example prior art portal displaying financial information.

FIG. **18** sets forth a screenshot of an example personalized portal.

DETAILED DESCRIPTION

FIG. **1** shows a diagram of a communications system which is suitable to practice the present invention. In the exemplary embodiment shown, a proxy server **30** is connected to a communication network through communication link **200** to a gateway **90**. The proxy **30** is shown shared by a number of clients **60** who are each executing an information retrieval program such as a browser. In an alternate embodiment, the proxy and the client browser can be executed as processes on the same client machine. Servers

70 provide information content to the clients 60 utilizing some document serving protocol, such as the Hypertext Transfer Protocol (HTTP) as described in T. Berners-Lee et al., "Hypertext Transfer Protocol—HTTP/1.0," RFC 1945, Network Working Group, 1996, which is incorporated by reference herein. As used herein, a document serving protocol is a communication protocol for the transfer of information between a client and a server. In accordance with such a protocol, a client 60 requests information from a server 70 by sending a request to the server and the server responds to the request by sending a document containing the requested information to the server. Servers, and the information stored therein, can be identified through an identification mechanism such as Uniform Resource Locators (URL), as described in detail in T. Berners-Lee et al., "Uniform Resource Locators," RFC 1738, Network Working Group, 1994, which is incorporated herein by reference. In an advantageous embodiment, the network is the Internet and the servers 70 are Web servers.

Proxy server 30, as is known the art, has a processor 140, memory 130 and a non-volatile storage device 150 such as a disk drive. The memory 130 includes areas for the storage of, for example, computer program code and data, as further described below. The proxy server 30 is shown executing two processes: an access server 40 and a 105 web server 50, both of which are further described in the pending utility patent application, "Information Access System And Method," Ser. No. 08/994,600, filed on Dec. 19, 1997, now abandoned,. The access server 40 behaves like a typical proxy server and accepts document requests (in a preferred embodiment, using standard TCP ports and HTTP) and routes them to other proxies or the desired server. The built-in server 50 is designed to act as a function execution engine rather than just an information provider.

FIG. 2 illustrates an exemplary system architecture, adapted for HTTP and implemented as Java classes, providing the functionality necessary to practice an embodiment of the present invention. When the proxy application is started, the main thread 2010 listens on the proxy port and receives and responds to HTTP requests from 115 clients 60 (browsers or other proxies). The system creates a new User Thread 2020 to serve each new request. The iserver class 2025 parses the client's request and forwards commands to iagent 2035 for remote web access or to ihttpd 2026 for localhost access. The iserver class 2025 also implements various protocol extensions further discussed below in Section 1. The ihttpd class 2026 implements the web server described above and will return an external file with the http header or execute a local CGI script to generate the replying message online. If the content is in a special scripting format described in further detail below in Section 3, icmd 2028 will be invoked to parse the script and interpret and execute any commands embedded in the document page.

The iagent class 2035 of the agent thread 2030 connects to a remote web server 70 or proxy to request a Web page. iagent 2035 can cache the page and return the page to iserver 2025 or to ihtwalk 2045, a facility to walk the html tree structure to collect and archive pages. The ihtwalk class facility is further described below in Section 4 in the description of the walking facilities.

1. Protocol Extensions

The URL and HTTP protocols can be extended to include additional system commands that advantageously permit conventional information retrieval clients to be utilized with embodiments of the present invention. This can be accomplished in a manner that is transparent to the client browser application and, thus, requires no new user interface.

As an illustration of an example format for a URL extension is:

URL?iproxy&command

where "?iproxy" is a keyword, and "command" is a service that might be applied to a given URL or a system command that is irrelevant to the given URL. For example, the command could invoke a archiving service (as described in further detail below in Section 5) or can be used to tailor system functions. The iserver class 2025 parses the commands and forwards the command to ihttpd 2026 which, in turn, forwards the command to icmd 2028 for processing. FIG. 3 sets forth a variety of system commands that can be defined using URL extensions.

In another embodiment of the present invention, a URL extension can be used to invoke a dynamically generated menu of entry points for available services and present it to the user's browser when requested. For example, the URL naming protocol can be extended by introducing a new naming scheme:

URL??

where the double-question mark "??" is a trigger to invoke a service menu. The service menu can be implemented to inherit the same cookies as the current URL. The proxy subsystem intercepts the request from the browser with the above request pattern, generates a menu on-the-fly and returns the menu to the browser as an HTML file. FIG. 4 shows an illustrative example of a menu obtained after a user browses a webpage, here "http://www.interactive.wsj.com/pages/techmain.html", and types "??" after the URL and presses enter. The menu platform defines entries a number of services and user-defined macros on a per URL-set basis. For example, FIG. 4 permits the user to, inter alia, change cache policies, archive or prefetch pages in the html tree, or search the page hierarchy for a keyword. The menu system chooses the proper menu description based on the current URL.

The menu service has numerous advantages. It does not introduce a new graphical user interface, but rather presents the menu in HTML content shown in normal browsers. It preserves user's cookies and is easily extensible. New service entries can be readily plugged into the menu. The concept of the service menu is akin to pushing special keys during a traditional telephone conversation, for example, "##"—the network then places the line on hold and announces a service menu for selection; after menu selection, the line is placed back to the original conversation.

New commands can also be introduced to HTTP for special communications among multiple proxy servers to, for example, establish special connections (like persistent channels) and/or perform value-added services (like TCP forwarding). For example, the commands set forth in FIG. 5 can be defined for such communications and used to extend the conventional HTTP commands.

2. Filters

Support is provided for the processing of data by filters. FIG. 6 illustrates how filters can be applied to http headers, pages returned from the web, and pages returned from a local cache. For example, an input filter can be used to add new components (menubars, etc.) or modify returned pages (replace some remote data with local data, etc.). Data from servers can also be condensed, compressed, encrypted, patched, etc., prior to the corresponding filters converting it back into its original format before returning it to a user.

The filter functions shown in FIG. 6 can be specified in a configuration file for the proxy's built-in web server. The function can be specified with a corresponding URL pattern, such that the filter is applied to URLs matching the pattern. For example, the following entries:

InputFilter /bin/fpack.cgi http://*/resume.html

OutputFilter /bin/funpack.cgi http://*/resume.html

HeaderFilter /bin/forward.cgi http://www.att.com/*

specify cgi functions for the indicated filters and URL patterns. As indicated in the HeaderFilter entry, all HTTP calls for the web server www.att.com will be processed by the cgi function forward.cgi before being sent out to the remote web server (or proxy server), and so on. Filters can also be specified using an extended URL command. For example, the system can be configured to calls filters with the arguments:

http://localhost/javabin/inputfilter?url=cached_url&path=cache_file

http://localhost/javabin/outputfilter?url=cached_url&path=cache_file

http://localhost/javabin/headerfilter?ht.header=header_lines

Filter programs can be implemented as cgi-bin programs and inherit the cgi-bin API. Every filter programs implemented as a Java class will create an instance when invoked. FIGS. 7A, 7B, and 7C show example cgi-bin filter programs. FIG. 7A shows an inputfilter; the Execute method reading the "cache_file" and updating the file if necessary. In FIG. 7B, outputfilter uses "cache_file" as input and sends the result to OutputStream which will be sent back to the original caller. FIG. 7C shows headerfilter which gets the original http header (multiple lines without the empty line) from Hashtable, and outputs a new header to OutputStream.

3. Scripting Facilities

In addition to Java class invocation mechanisms and standard scripting mechanisms such as Ksh and Perl, it is advantageous to provide support for document pre-processor scripting. Proxy-side scripting allows the different system components to be integrated in a light-weight manner that has a syntax and a semantics that is very HTML-like. Scripting provides a method for plugged-in services/functions to access built-in functions, server status and data structure, and the cache. A scripting proxy pre-processor can also provide a communication media for multiple proxy servers.

The inventors devised a scripting language providing extra macros statements based on the standard document markup language HTML. The proxy server pre-processes the scripts and turns the comments into pure HTML. In a preferred embodiment of such scripting, it should include support for variable declarations, conditional statements, sets of built-in functions, and interfaces to invoke cgi-bin. The script can be a standard HTML-like plain text file that contains the macro statements. In order to identify the document as a script, an identifier should be included; for example, the first line in the script can start with:

#!/iproxy/script

The arguments of a cgi-bin can be accessed using "${arg}" inside the script file. FIG. 8 sets forth a list of built-in statements that are useful for scripting. When using the ":block" and ":javabin" commands, the script will invoke a javabin class with the arguments set forth in FIG. 9.

4. Walking Facilities

One useful facility is a basic function supporting a mechanism to walk through document page hierarchies. This is similar to what "find" and "tw" (file tree walk) do on Unix file systems. The walking action is specified by a root URL where the action starts, a specification of how many levels the action will visit, and certain additional properties such as whether or not image files should be included. For each visited page, one or more of a list of functions can be invoked one by one to perform tasks on a cache of the page. Examples of such functions include functions for archiving the web pages, searching for keywords, and creating index tables.

See FIG. 10 for the structure of a walking function, "htwalk", designed in accordance with a preferred embodiment of the present invention. The syntax for the walking function is given in FIG. 3. For example, for the URL

http://www.att.com/?iproxy&htwalk=3,-local,-image,archive,grep=Cable

the system walks down 3 levels for pages referred (directly or indirectly) by www.att.com, including image files, but only for those pages on the local server. For each visited page, the system calls the function archive to archive the page and the function grep to search for the keyword "Cable" in the page. For each visited page, the system calls the following cgi-bin programs one-by-one:

/bin/archive.cgi?url=visited_page&args=int_no,-local&path=cached_file
&htwalk=archive

/bin/grep.cgi?url=visited_page&args=int_no,-local&path=cached_file
&htwalk=grep&walkopt=Cache

FIG. 11 sets forth an example of a grep class used to implement the walking function.

5. Archiving Service

The URLs processed by the proxy **30** can be extended to include archive directives that add data to a storage repository, e.g. device **150**, and for retrieving the archived data. The new commands are intercepted and performed by the proxy server. Because a proxy is used as a middleman between the browser and the web servers, the new archiving services are just plug-in components and do not interfere with existing components and protocols.

As an advantageous example, the URL naming scheme can be extended to include URLs in the following format:

http://view@host/path

where the "view" is a date, for example in the format yyyymmdd, when the corresponding page (i.e., http://host/path) has been retrieved from the original web server and stored into the archive repository. The timestamp can be used as the key to locate the page from the repository. For example,

http://19980701@www.att.com/would refer to the page http://www.att.com that was retrieved and archived from the www.att.com web server on Jul. 1, 1998.

More advanced features can be implemented to allow specifying an action in front of the date for locating an alternative page on the archive server if the dated page does not exist. The system first locates the page archived on that date. If the desired page is not found in the repository, the system then searches for the page in the repository before or after the date and returns the first found page. For example, the following syntax for naming archived pages can be used:

http://[+|-]yyyymmdd[.hhmmss]@host/path

Thus, http://+date@host/path would choose the first page found after the specified date; on the other hand, http://-date@host/path points to the first page before the date. For example, http://+19980701@www.att.com points to the first page that was archived on Jul. 1, 1998 or after, while http://19980701@www.att.com is for the page archived on Jul. 1, 1998, or before. As shown in the syntax, the granularity of "view" can be readily taken down to the second level of hours, minutes, and seconds.

The above naming scheme advantageously is compatible with the conventional URL protocol, which defines network resource naming as

proto://[user[:password]@]host/path. For example, the URL

ftp://foo:bar@ftp.research.att.com/README points to a README page on the FTP server ftp.research.att.com, while accessing on behalf of the user foo with the password bar. The user/password portion of the URL protocol is undefined in the HTTP protocol; accordingly, the above naming scheme can take advantage of this for archive naming.

Multiple methods can be used to invoke the archive service and store data in the repository:

Command Extensions. Using any of the methods described above under Section 1, a user can invoke the archiving service. For example, the following URL:

http://www.research.att.com/iPROXY.html?iproxy&action=archive

can be used to cause the system to archive the page

http://www.research.att.com/iPROXY.html.

Scheduled Archiving. The system can also be extended to contain a server that executes an archiving task at a designated time, e.g. like cron command in Unix systems. The walking facilities described above can be used to "walk" through a web site for a set of HTML pages. As described above, the walking can be defined by a root URL and parameterized by (a) the depth of walking through hyper-references under HTML pages, (b) with or without image files embedded in HTML pages, and (c) walking through pages on the local web site or on all web sites. FIG. 12 shows an example page interface for a cron cgi program which can be used to add a cron job to archive a web site. Users can schedule an archive task on a daily or weekly basis.

Transparent Archiving. The system can support a finction of archiving selected web pages whenever they have been accessed automatically. The specification can be done through the CacheFlag of a cache command, which specifies the caching policy:

CacheFlag Archive URL-expression

Whenever a requested URL matches the URL-expression, the system puts the data in the archive repository.

Archiving Browser Cache. The system can support a function for scanning data cached on a browser's cache area and archiving them into the repository.

As for retrieving information from the repository, FIG. 13 shows how an example screenshot of an interface that can be used to browse the archived information. Each month's data of each website can be stored in a repository structure similar to a Unix-like pax archive. A cgi program creates a page on-the-fly based on the contents of the archive and creates hyperlinks for each website name. The hyperlinks are listed for each month when the certain pages of the website were archived. For example, the AT&T website has a cron job to archive the web pages on a daily basis, so the data in FIG. 13 has all twelve months in 1998 listed as hyperlinks.

Multiple archive repositories can take advantage of the above-mentioned features for inter-proxy communication to share the burden of storing the information. For example, FIG. 14 shows how multiple archive repositories can collaborate in the archiving task. Proxy server in FIG. 14 queries another repository in order to answer a user request. As shown in FIG. 14, the following advantageous web interface can be used to search the repository or repositories:

http://proxyhost/bin/iget/view@host/path

A repository search path variable can be set using a command such as:

RepositoryPath=Host:Port[;Host:Port]*

Thus, multiple repositories may be easily addressed and accessed to quickly locate the desired information.

6. Packaging Service

A packaging service can be provided which would allow the browsing of document pages from portable packages. The package is portable in that it can be possible to attach one to an e-mail or to copy one from one machine to another. To create a package, the walking facility may be used to visit the sets of web pages that are designated to be packed. Thus, the system would accept (a) a root URL, (b) a parameter expressing the depth of the walking through the root URL's direct and indirect references, (c) an image option to decide whether or not to include the image files when walking through the web pages, (d) a reference filter option (for accessing all references or filtering out some of them), (e) mechanisms for storing cookies needed for accessing the root URL; and (f) a package name. The system then walks through the set of web pages rooted by the designated URL and packs them into a package using the designated name. The package can use any of the number of known conventional compression formats to package the data. The package is advantageously self-contained, including two physical files: one for contents and another for indexing. Each package can maintain its own index table.

In order to browse the package, the proxy system can support functions to access the web pages stored in the package as if they were from the original web sites. The packages are stored on a local disk, and browsing does not require a network connection. When browsing, the server uses the above-mentioned index table to locate corresponding content in the package. The server that generates the package may be different from the one that browses the package. The cookies used to traverse the web pages are automatically handled by the system.

New documents can easily be appended to an existing package. A single package may, in fact, contain more than one root URL. Each root URL in the package can be chosen as an entry point to browse the package.

7. Personalized Services

It is notable that the proxy system described above can have access to a user's private information such as the web access history and sensitive financial information. A user may not be comfortable with providing this information to a server across the communication network. The same information can be stored in a safe location locally, on the client machine where the proxy has been integrated on the client-side or on a local centralized proxy closer to the user. The present invention permits several new personalized services to be provided.

The proxy can be used to integrate in an automatic manner the information provided by a typical portal with sensitive personal information. For example, a user issues a request for the following URL:

http://www.att.net/?iproxy&action=portal

The portal command would cause the proxy server to retrieve the home page from www.att.net, which has been encoded with scripting directives that instruct the proxy server how to process the local data and merge it with the server contents. It then presents the personal portal page back to the user. In order to provide scripting that is non-intrusive to other users who are not using the present invention, the scripting directives can be embedded in HTML comments as shown in FIG. 15. The proxy server intercepts the directives in FIG. 15 and performs the necessary actions before returning the server portal page to the browser. All the directives, being embedded in an HTML comment, are ignored by browsers not using a proxy server configured in accordance with the present invention.

The following are some illustrative examples of how to use the capabilities of personalizing information retrieved from the public servers. They are merely examples and are not meant to limit the nature of the present invention:

Personal Web Page Reminders and Hot Sites. Since the proxy server can log a user's web accesses, it can analyze the log and use the data to provide new web services that can improve the user's browsing experience. FIG. **16** shows, for example, two possible services: (1) "TO-READ" homepages. A user can specify a list of websites and corresponding frequencies when certain websites should be visited. The proxy can check the last visiting dates and schedule a list of pages that the user should visit today. (2) "HOT Sites." The proxy server can compute the number of visits to each website and list the top ten websites with their last visiting dates accordingly as soon as the user accesses his/her personal portal. There are several advantages of having such a list. The user may want to know the last visiting time of a favorite site—the timestamp can be displayed along with the website link. The user may wish to access the latest version by clicking on the link. The user may wish to compare the new version with the old using some differencing tool, especially since the previous versions of the web page can be archived as described above. FIG. **15** shows the directives used to construct the page displayed in FIG. **16**. The directive "to-read" constructs a list of web pages scheduled to be read; the directive "dolog" analyzes the current web access log to produce the statistics need for the next directive, "top10" which presents the results on the personal portal.

Personalized Financial Page. Most portals currently allow users to specify the stocks that they are interested in and display the latest stock price when a user accesses the personalized page. See FIG. **17** which shows a typical AT&T WorldNet portal showing various stock quotes. However, the portal cannot compute your current balance or net gain/loss unless you provide private and sensitive information like how many shares you own and when you purchased them. Most users would not like to provide such information to their portal page server. In accordance with an embodiment of the present invention, such personalized information such as real purchase price, commission fees, and the number of shares of each stock can be stored and accessed by the proxy server. By constructing an output filter for the stock page, the proxy server can retrieve the private information, combine it with stock quotes provided by the portal site to compute the balance, net gain/loss, and other interesting personalized information/statistics. FIG. **18** shows the same view as FIG. **17** personalized with the user's information. This can be accomplished by a specification like the following in the proxy configuration file:

```
OutputFilter/bin/portfolio.cgihttp://
    stocks.planetdirect.com/portfolio.asp
```

This instructs the system to apply the Java class "portfolio" as an output filter whenever the browser issues the corresponding http request. The numbers in FIG. **18** are visible only to the client and not the original portal server. The user is shown as having bought 267 AT&T shares, 50 Netscape shares, and 40 E*Trade shares at the respective prices of $36.50, $21.0, and $39.38, each. The commissions were 0, $19.95, and $19.95. The total gain was $17,799.02. The numbers replaced by the proxy server are shown in a different shade.

Personal Web Archive. While current search engines allow users to find pages of a certain topic easily, they do not offer much help in looking and viewing the pages a user has seen in the past, except for those that are still kept in the browser cache. Due to the sharp decrease in storage costs, a client-side proxy can afford to archive all the web pages a user has seen so that any of these pages can be retrieved easily later on—without even bookmarking them. Existing webpage search tools such as Alta Vista Discovery can be used to index the web archive and search the pages. A user can then quickly conduct a search of all pages he/she has seen in the last year, for example. As described in the archiving section above, the instant system can intercept http requests and effectively extend the URL name space to address pages stored in the archive by adding a timestamp in front of the regular http address. Even as the web pages go through major redesigns, the original pages can be accessed using the archive extensions to give the same content.

The foregoing Detailed Description is to be understood as being in every respect illustrative and exemplary, but not restrictive, and the scope of the invention disclosed herein is not to be determined from the Detailed Description, but rather from the claims as interpreted according to the full breadth permitted by the patent laws. It is to be understood that the embodiments shown and described herein are only illustrative of the principles of the present invention and that various modifications may be implemented by those skilled in the art without departing from the scope and spirit of the invention. For example, the detailed description has been described with particular emphasis on the Internet standards of HTFP, URL's, and HTML. However, the principles of the present invention could be extended to other protocols for serving information from a server to a client. Such an extension could be readily implemented by one of ordinary skill in the art given the above disclosure.

What is claimed is:

1. A proxy server interposed between at least one client browser and a server, the proxy server comprising:

a first interface for establishing a document serving protocol channel to the server;

a second interface for establishing a second document serving protocol channel to the client browser;

a storage device for archiving documents;

a processor adapted to

(i) receive a first requests from the client browser for one or more documents on the server,

(ii) retrieve the documents from the server,

(iii) store the documents on the storage device,

(iv) associate the stored documents with date information indicating when the documents were retrieved from the server,

(v) sending the documents to the client browser,

(vi) receive at a later time a second request from the client browser wherein the request contains an archive directive,

(vii) parse the second requests for an archive directive, and,

(viii) where the requests contain an archive directive requesting an archived copy of the one or more documents and specifying an archival date or a range of archival dates,

(a) access the storage devices,

(b) search for date information associated with documents stored on the storage device that match the archival date specified in the request,

(c) if there is matching date information, retrieving the documents from the storage device associated with the date information and sending the retrieved documents to the client browser, and

(d) if there is more than one matching date information, sending a list of archived documents

with associated matching date information to the client browser.

2. The proxy server of claim 1 wherein the documents stored on the storage device are stored as a set in a portable package on the storage device and are sent to the client browser as a set in the portable package when requested by the client browser in the archive directive in the request.

3. The proxy server of claim 1 wherein the archive directive is expressed as a uniform resource locator with a format of date@host/path.

4. The proxy server of claim 1 wherein the document serving protocol is the hypertext transfer protocol.

5. The proxy server of claim 1 wherein the document is a web page.

6. A computer readable medium containing executable program instructions for performing a method on a proxy server interposed between at least one client browser and a server comprising the steps of:

receiving a first requests from the client browser for one or more documents on the server,

retrieve the documents from the server,

store the documents on a storage device,

associate the stored documents with date information indicating when the documents were retrieved from the server,

sending the documents to the client browser,

receive at a later time a second request from the client browser wherein the request contains an archive directive,

parsing the second requests for an archive directive, and,

where the requests contain an archive directive requesting an archived copy of the one or more documents and specifying an archival date or a range of archival dates, accessing the storage device,

search for date information associated with documents stored on the storage device that match the archival date specified in the request,

if there is matching date information, retrieving the documents from the storage device associated with the date information and sending the retrieved documents to the client browser, and

if there is more than one matching date information, sending a list of archived documents with associated matching date information to the client browser.

7. The computer readable medium of claim 6 wherein the documents stored on the storage device are stored as a set in a portable package on the storage device and are sent to the client browser as a set in the portable package when requested by the client browser in the archive directive in the request.

8. The computer readable medium of claim 6 wherein the archive directive is expressed as a uniform resource locator with a format of date@host/path.

9. The computer readable medium of claim 6 wherein the document serving protocol is the hypertext transfer protocol.

10. The computer readable medium of claim 6 wherein the document is a web page.

* * * * *

US006643696B2

(12) **United States Patent**
Davis et al.

(10) **Patent No.: US 6,643,696 B2**
(45) **Date of Patent: Nov. 4, 2003**

(54) **METHOD AND APPARATUS FOR TRACKING CLIENT INTERACTION WITH A NETWORK RESOURCE AND CREATING CLIENT PROFILES AND RESOURCE DATABASE**

(76) Inventors: **Owen Davis**, 214 W. 102nd St., New York, NY (US) 10025; **Vidyut Jain**, 352 6th Ave., Brooklyn, NY (US) 11215

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/210,235**

(22) Filed: **Dec. 11, 1998**

(65) **Prior Publication Data**

US 2002/0099812 A1 Jul. 25, 2002

Related U.S. Application Data

(63) Continuation-in-part of application No. 09/120,376, filed on Jul. 21, 1998, now Pat. No. 6,138,155, which is a continuation of application No. 08/821,534, filed on Mar. 21, 1997, now Pat. No. 5,796,952.

(51) **Int. Cl.**[7] **G06F 13/00**
(52) **U.S. Cl.** **709/224**
(58) **Field of Search** 705/7, 30, 10; 707/526, 104; 709/200, 202, 203, 213, 217, 218, 219, 223, 224

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,977,594	A *	12/1990	Shear	705/53
5,638,443	A *	6/1997	Stefik et al.	705/54
5,675,510	A *	10/1997	Coffey et al.	709/224
5,682,525	A *	10/1997	Bouve et al.	705/54
5,706,502	A *	1/1998	Foley et al.	707/10
5,708,780	A *	1/1998	Levergood et al.	709/229
5,710,918	A *	1/1998	Lagarde et al.	707/10
5,715,453	A *	2/1998	Stewart	707/104
6,108,637	A *	8/2000	Blumenau	705/1

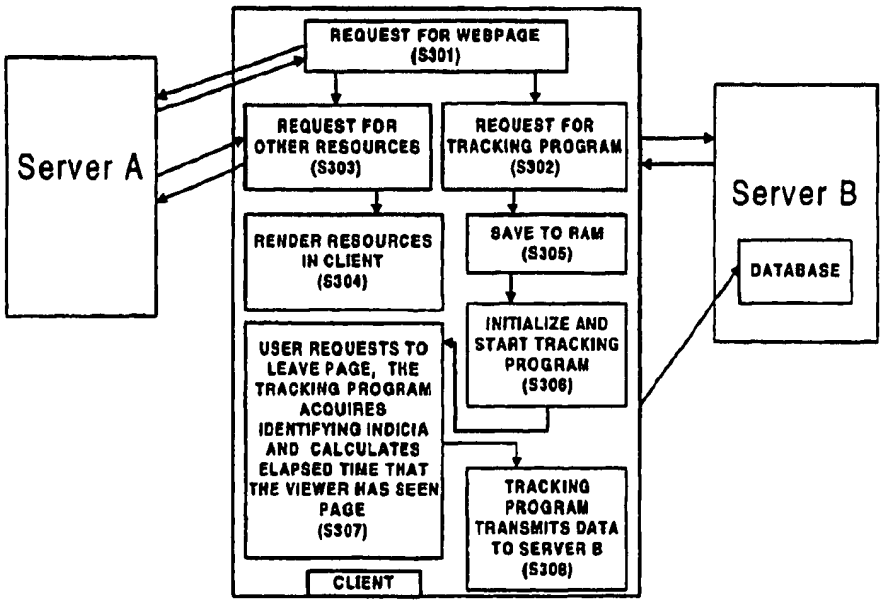
* cited by examiner

Primary Examiner—Robert B. Harrell
(74) *Attorney, Agent, or Firm*—Brown, Raysman, Millstein, Felder & Steiner LLP

(57) **ABSTRACT**

A method for monitoring client interaction with a resource downloaded from a server in a computer network includes the steps of using a client to specify an address of a resource located on a first server, downloading a file corresponding to the resource from the first server in response to specification of the address, using the client to specify an address of a first executable program located on a second server, the address of the first executable program being embedded in the file downloaded from the first server, the first executable program including a software timer for monitoring the amount of time the client spends interacting with and displaying the file downloaded from the first server, downloading the first executable program from the second server to run on the client so as to determine the amount of time the client interacts with the file downloaded from the first server, using a server to acquire client identifying indicia from the client, and uploading the amount of time determined by the first executable program to a third server.

59 Claims, 11 Drawing Sheets



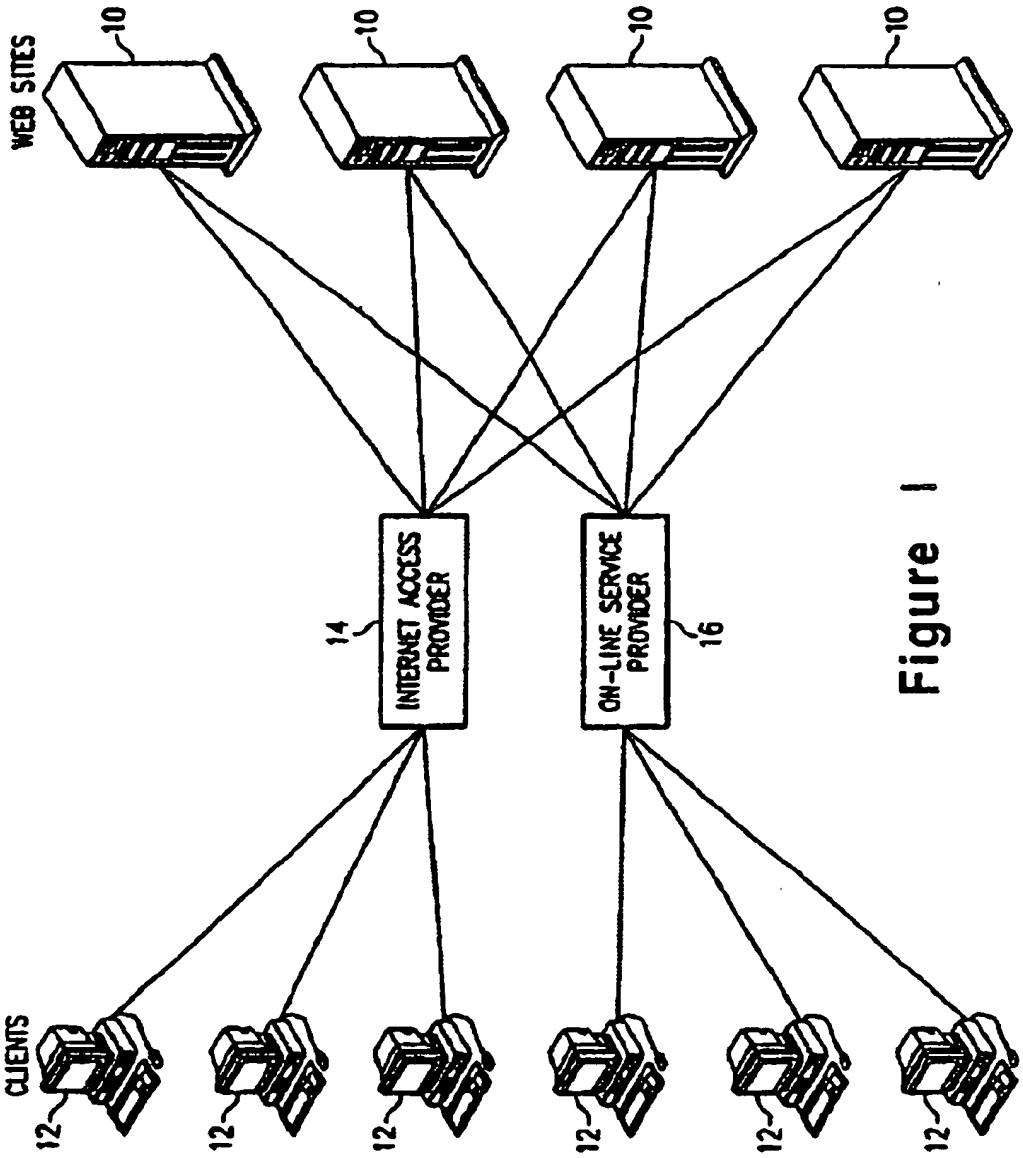
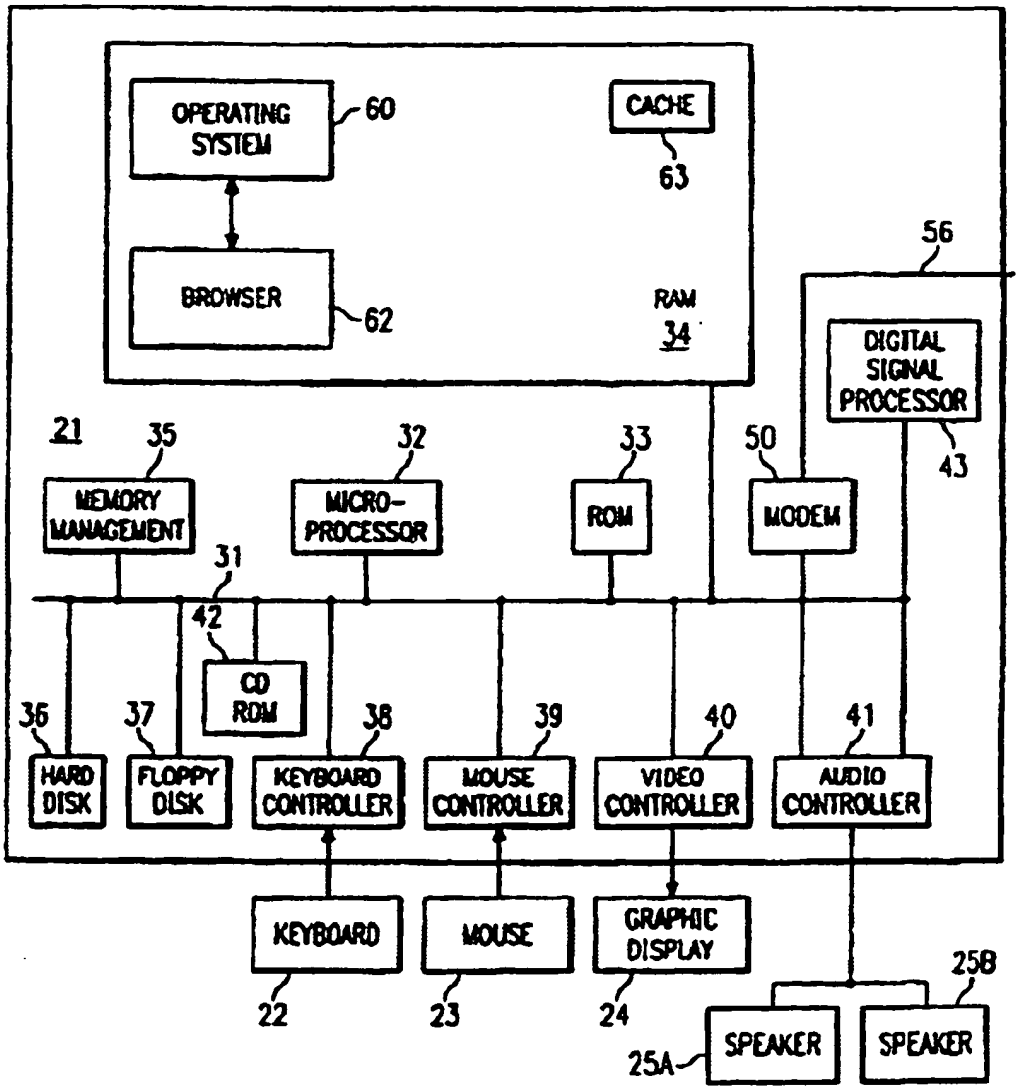

WEB SITES
10
10
10
10
INTERNET ACCESS PROVIDER
14
ON-LINE SERVICE PROVIDER
16
CLIENTS
12
12
12
12
12
12

Figure 1

Figure 2


OPERATING SYSTEM
60
CACHE
63
BROWSER
62
RAM
34
56
DIGITAL SIGNAL PROCESSOR
43
21
35
MEMORY MANAGEMENT
32
MICRO-PROCESSOR
33
ROM
50
MODEM
31
42
CD ROM
36
HARD DISK
37
FLOPPY DISK
38
KEYBOARD CONTROLLER
39
MOUSE CONTROLLER
40
VIDEO CONTROLLER
41
AUDIO CONTROLLER
KEYBOARD
22
MOUSE
23
GRAPHIC DISPLAY
24
25B
25A
SPEAKER
SPEAKER

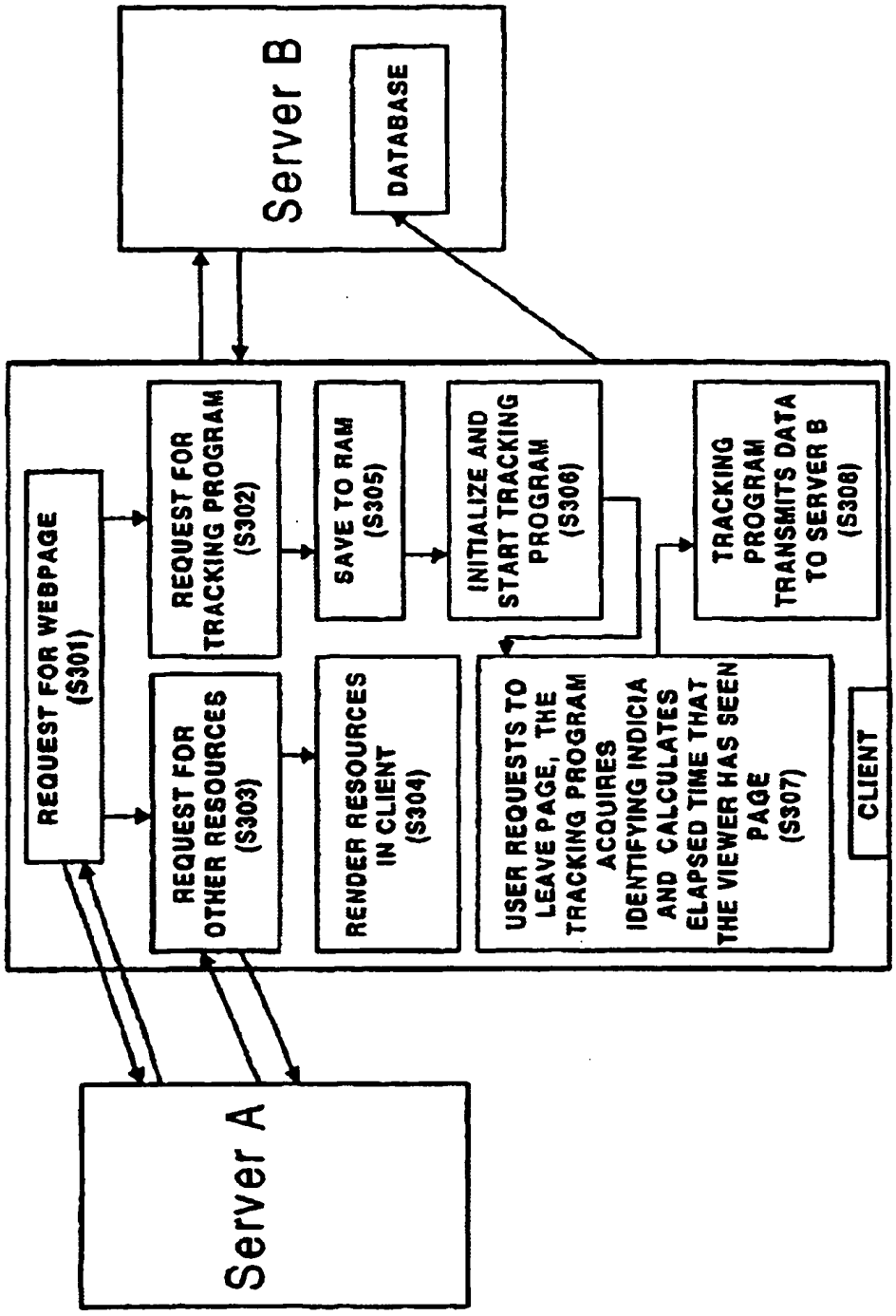

Server B
DATABASE
REQUEST FOR WEBPAGE (S301)
REQUEST FOR TRACKING PROGRAM (S302)
SAVE TO RAM (S305)
INITIALIZE AND START TRACKING PROGRAM (S306)
TRACKING PROGRAM TRANSMITS DATA TO SERVER B (S308)
REQUEST FOR OTHER RESOURCES (S303)
RENDER RESOURCES IN CLIENT (S304)
USER REQUESTS TO LEAVE PAGE, THE TRACKING PROGRAM ACQUIRES IDENTIFYING INDICIA AND CALCULATES ELAPSED TIME THAT THE VIEWER HAS SEEN PAGE (S307)
CLIENT
Server A

Figure 3

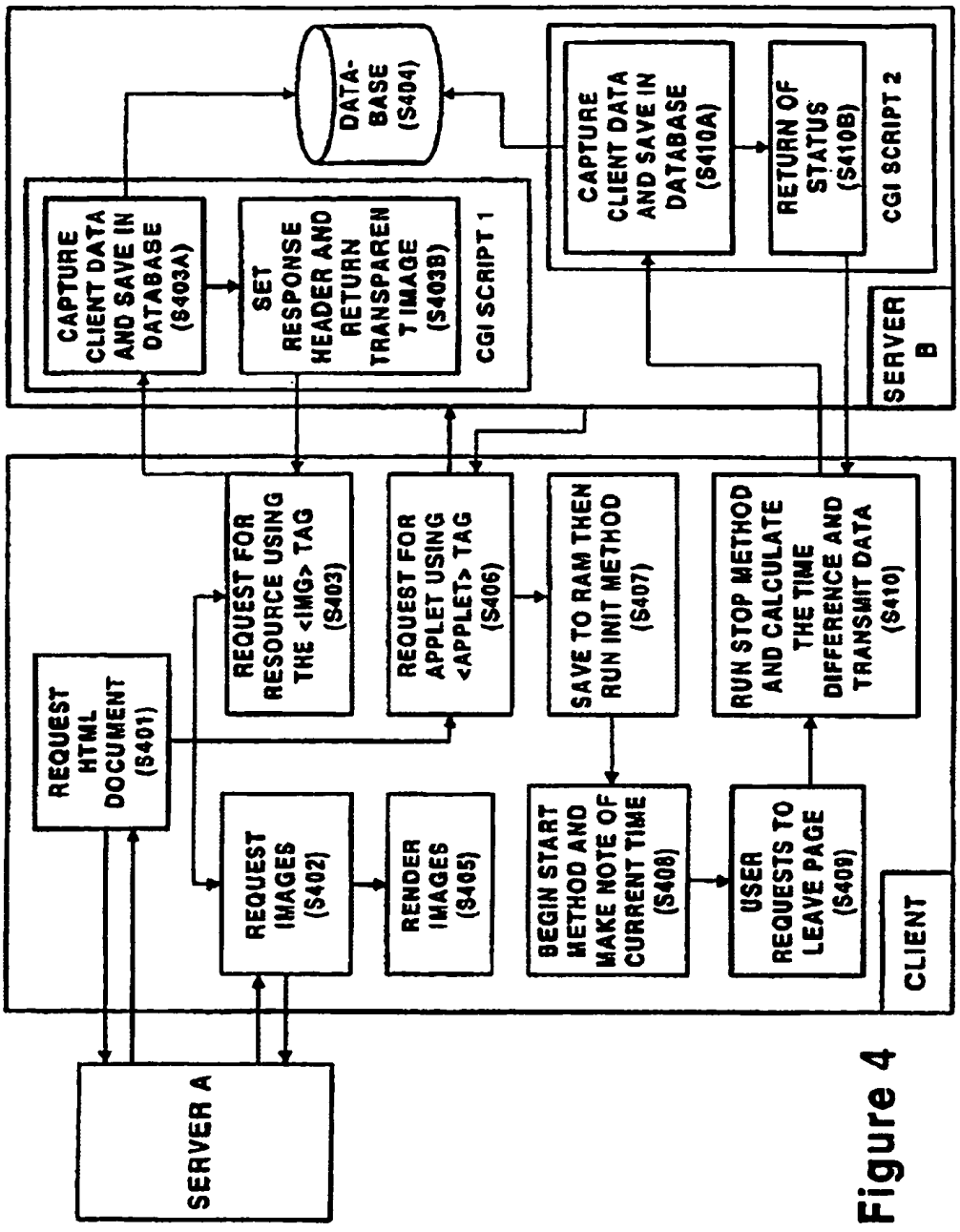

DATA-BASE (S404)
CAPTURE CLIENT DATA AND SAVE IN DATABASE (S403A)
SET RESPONSE HEADER AND RETURN TRANSPARENT IMAGE (S403B)
CGI SCRIPT 1
CAPTURE CLIENT DATA AND SAVE IN DATABASE (S410A)
RETURN OF STATUS (S410B)
CGI SCRIPT 2
SERVER B
REQUEST HTML DOCUMENT (S401)
REQUEST FOR RESOURCE USING THE <IMG> TAG (S403)
REQUEST FOR APPLET USING <APPLET> TAG (S406)
SAVE TO RAM THEN RUN INIT METHOD (S407)
RUN STOP METHOD AND CALCULATE THE TIME DIFFERENCE AND TRANSMIT DATA (S410)
REQUEST IMAGES (S402)
RENDER IMAGES (S405)
BEGIN START METHOD AND MAKE NOTE OF CURRENT TIME (S408)
USER REQUESTS TO LEAVE PAGE (S409)
CLIENT
SERVER A

Figure 4

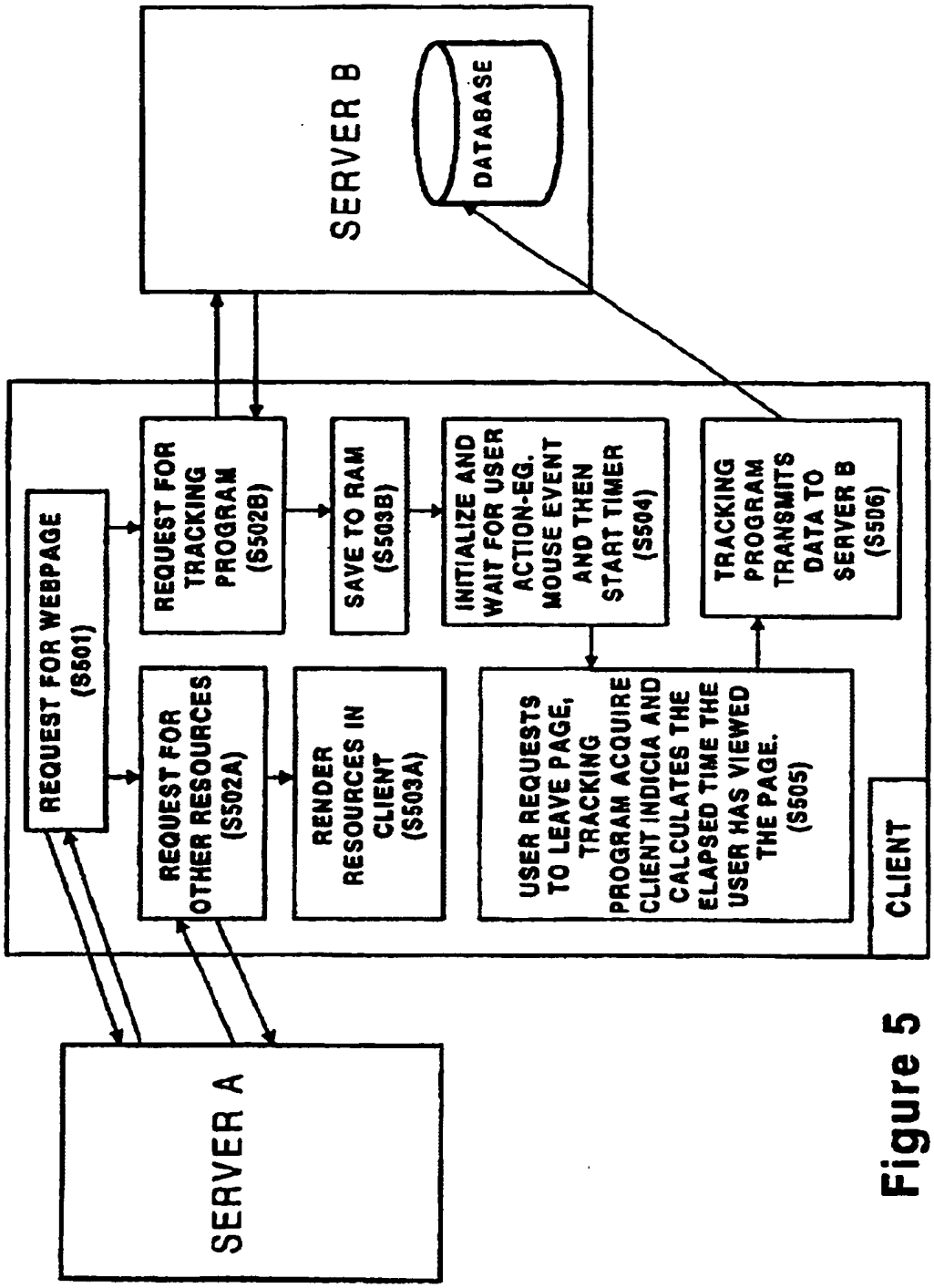

SERVER B
DATABASE
REQUEST FOR WEBPAGE (S501)
REQUEST FOR TRACKING PROGRAM (S502B)
SAVE TO RAM (S503B)
INITIALIZE AND WAIT FOR USER ACTION-EG. MOUSE EVENT AND THEN START TIMER (S504)
TRACKING PROGRAM TRANSMITS DATA TO SERVER B (S506)
REQUEST FOR OTHER RESOURCES (S502A)
RENDER RESOURCES IN CLIENT (S503A)
USER REQUESTS TO LEAVE PAGE, TRACKING PROGRAM ACQUIRE CLIENT INDICIA AND CALCULATES THE ELAPSED TIME THE USER HAS VIEWED THE PAGE. (S505)
CLIENT
SERVER A

Figure 5

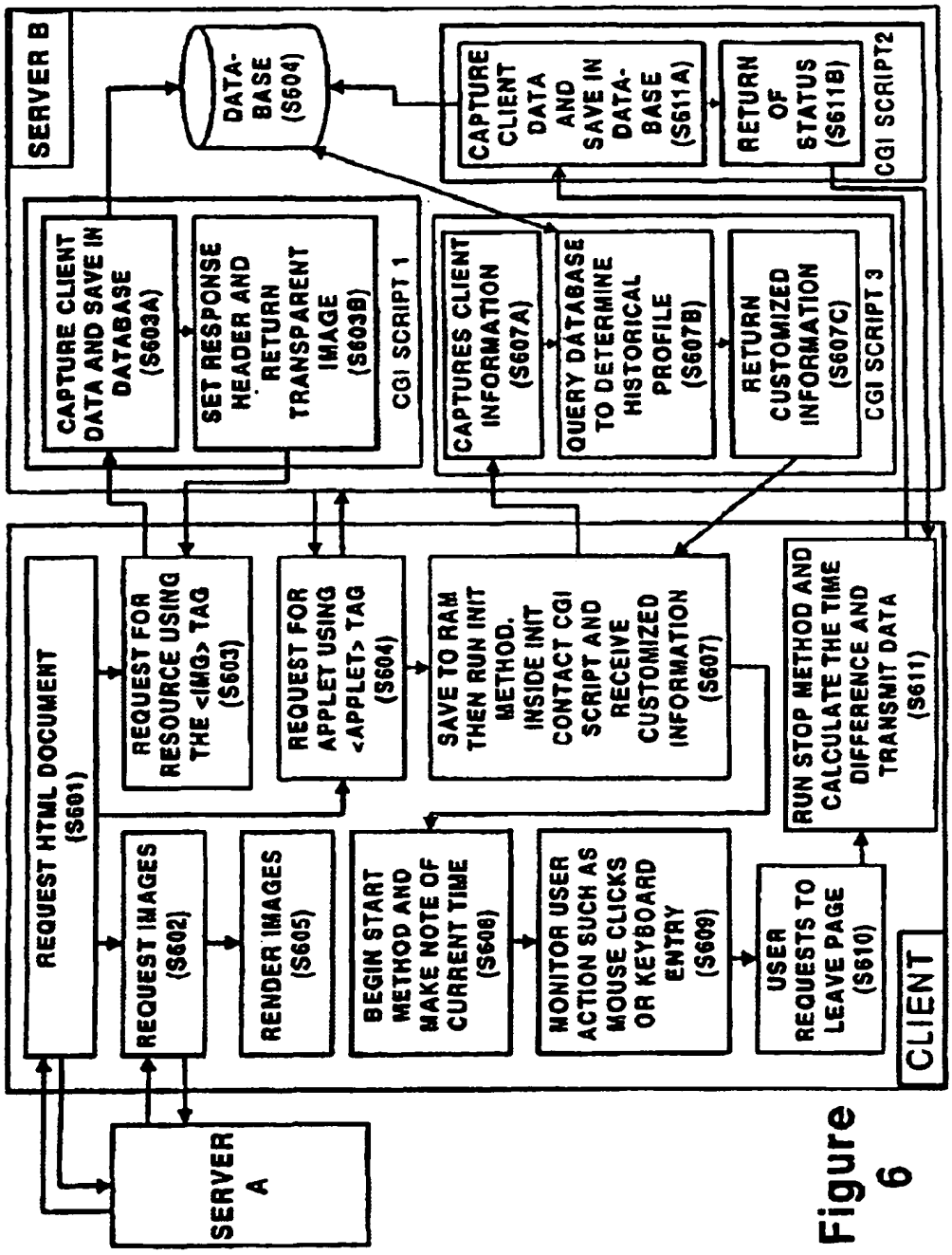

SERVER B
DATA-BASE (S604)
CAPTURE CLIENT DATA AND SAVE IN DATA-BASE (S611A)
RETURN OF STATUS (S611B)
CGI SCRIPT2
CAPTURE CLIENT DATA AND SAVE IN DATABASE (S603A)
SET RESPONSE HEADER AND RETURN TRANSPARENT IMAGE (S603B)
CGI SCRIPT 1
CAPTURES CLIENT INFORMATION (S607A)
QUERY DATABASE TO DETERMINE HISTORICAL PROFILE (S607B)
RETURN CUSTOMIZED INFORMATION (S607C)
CGI SCRIPT 3
REQUEST HTML DOCUMENT (S601)
REQUEST FOR RESOURCE USING THE <IMG> TAG (S603)
REQUEST FOR APPLET USING <APPLET> TAG (S604)
SAVE TO RAM THEN RUN INIT METHOD. INSIDE INIT CONTACT CGI SCRIPT AND RECEIVE CUSTOMIZED INFORMATION (S607)
RUN STOP METHOD AND CALCULATE THE TIME DIFFERENCE AND TRANSMIT DATA (S611)
REQUEST IMAGES (S602)
RENDER IMAGES (S605)
BEGIN START METHOD AND MAKE NOTE OF CURRENT TIME (S606)
MONITOR USER ACTION SUCH AS MOUSE CLICKS OR KEYBOARD ENTRY (S609)
USER REQUESTS TO LEAVE PAGE (S610)
CLIENT
SERVER A

Figure 6

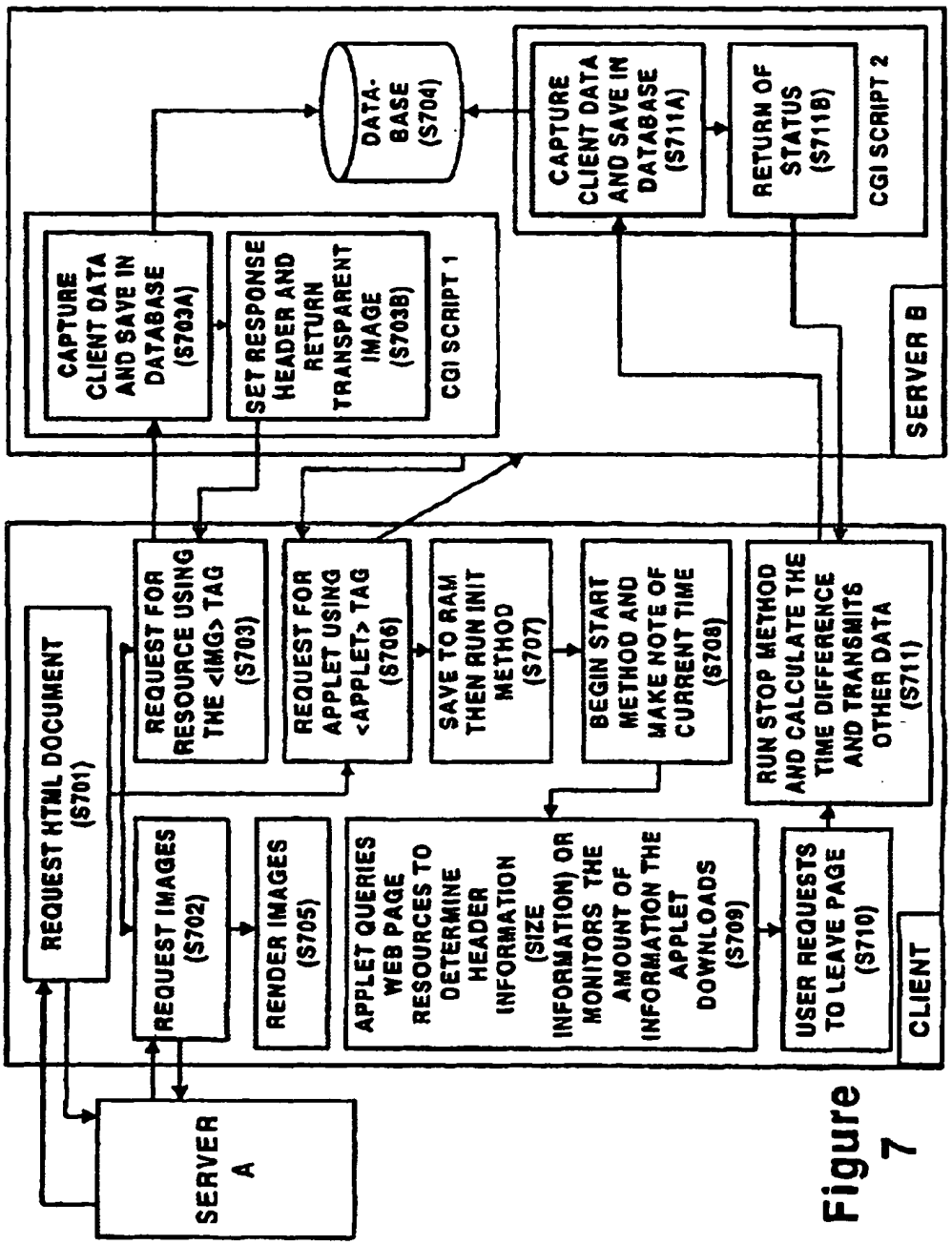

SERVER A
CLIENT
REQUEST HTML DOCUMENT (S701)
REQUEST IMAGES (S702)
RENDER IMAGES (S705)
REQUEST FOR RESOURCE USING THE <IMG> TAG (S703)
REQUEST FOR APPLET USING <APPLET> TAG (S706)
SAVE TO RAM THEN RUN INIT METHOD (S707)
BEGIN START METHOD AND MAKE NOTE OF CURRENT TIME (S708)
APPLET QUERIES WEB PAGE RESOURCES TO DETERMINE HEADER INFORMATION (SIZE INFORMATION) OR MONITORS THE AMOUNT OF INFORMATION THE APPLET DOWNLOADS (S709)
USER REQUESTS TO LEAVE PAGE (S710)
RUN STOP METHOD AND CALCULATE THE TIME DIFFERENCE AND TRANSMITS OTHER DATA (S711)
SERVER B
CAPTURE CLIENT DATA AND SAVE IN DATABASE (S703A)
SET RESPONSE HEADER AND RETURN TRANSPARENT IMAGE (S703B)
CGI SCRIPT 1
DATA-BASE (S704)
CAPTURE CLIENT DATA AND SAVE IN DATABASE (S711A)
RETURN OF STATUS (S711B)
CGI SCRIPT 2

Figure 7

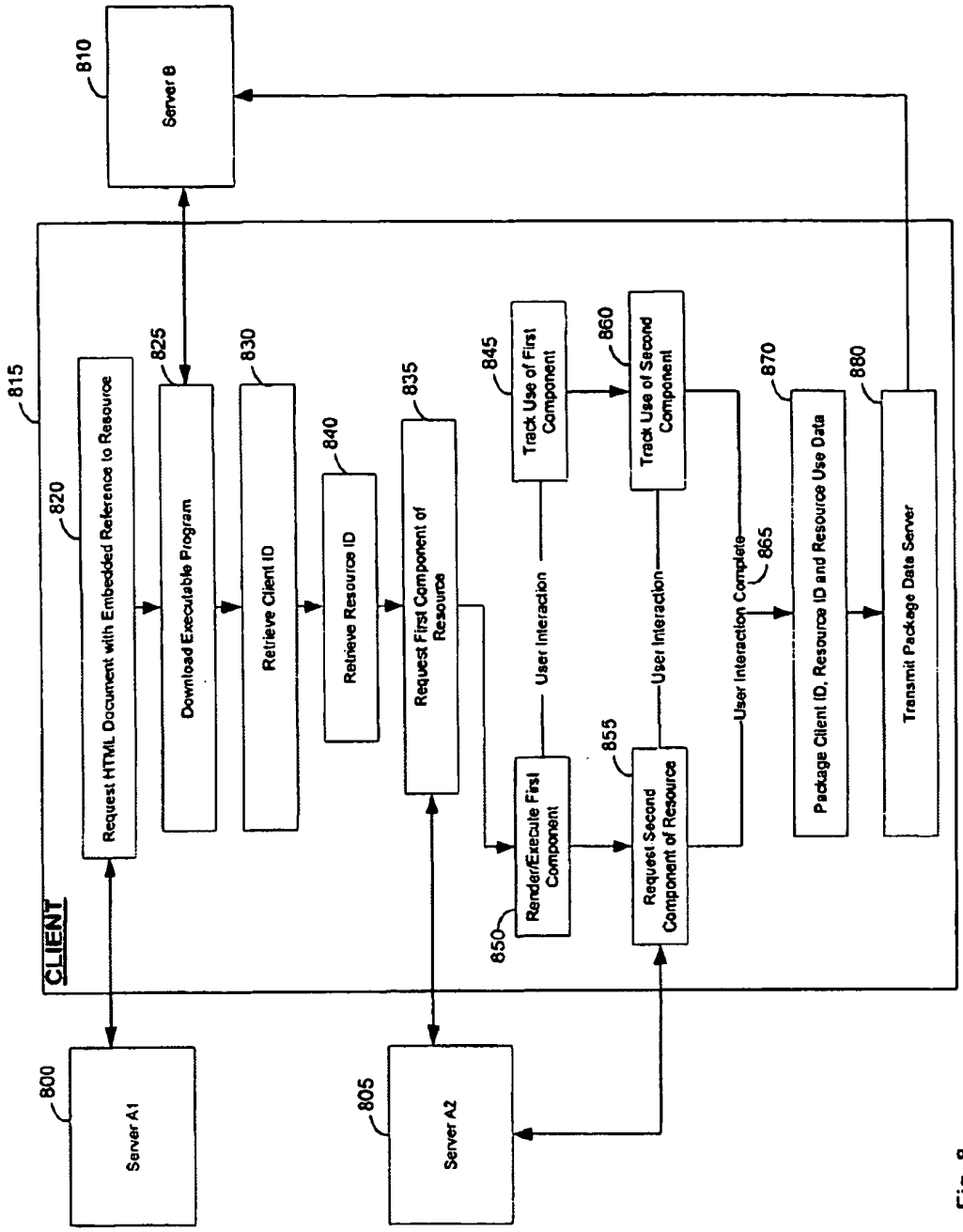

Server B
810
CLIENT
815
Request HTML Document with Embedded Reference to Resource
820
Download Executable Program
825
Retrieve Client ID
830
Retrieve Resource ID
840
Request First Component of Resource
835
Track Use of First Component
845
Render/Execute First Component
850
User Interaction
Track Use of Second Component
860
Request Second Component of Resource
855
User Interaction
User Interaction Complete
865
Package Client ID, Resource ID and Resource Use Data
870
Transmit Package Data Server
880
Server A1
800
Server A2
805

Fig. 8

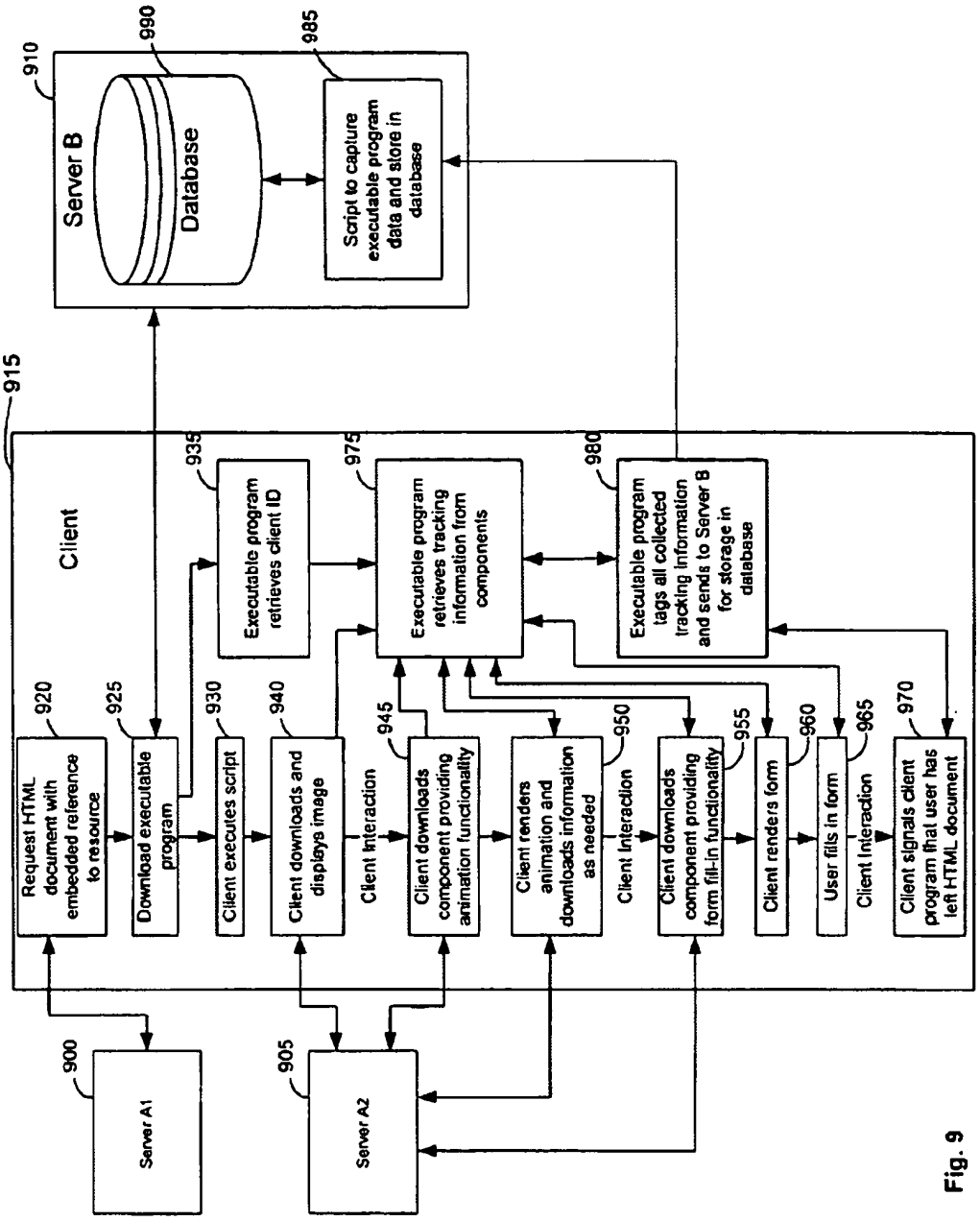

910
990
985
Server B
Database
Script to capture executable program data and store in database
915
Client
935
Executable program retrieves client ID
975
Executable program retrieves tracking information from components
980
Executable program tags all collected tracking information and sends to Server B for storage in database
920
Request HTML document with embedded reference to resource
925
Download executable program
930
Client executes script
940
Client downloads and displays image
945
Client interaction
Client downloads component providing animation functionality
950
Client renders animation and downloads information as needed
Client interaction
955
Client downloads component providing form fill-in functionality
960
Client renders form
965
User fills in form
Client interaction
970
Client signals client program that user has left HTML document
900
Server A1
905
Server A2

Fig. 9

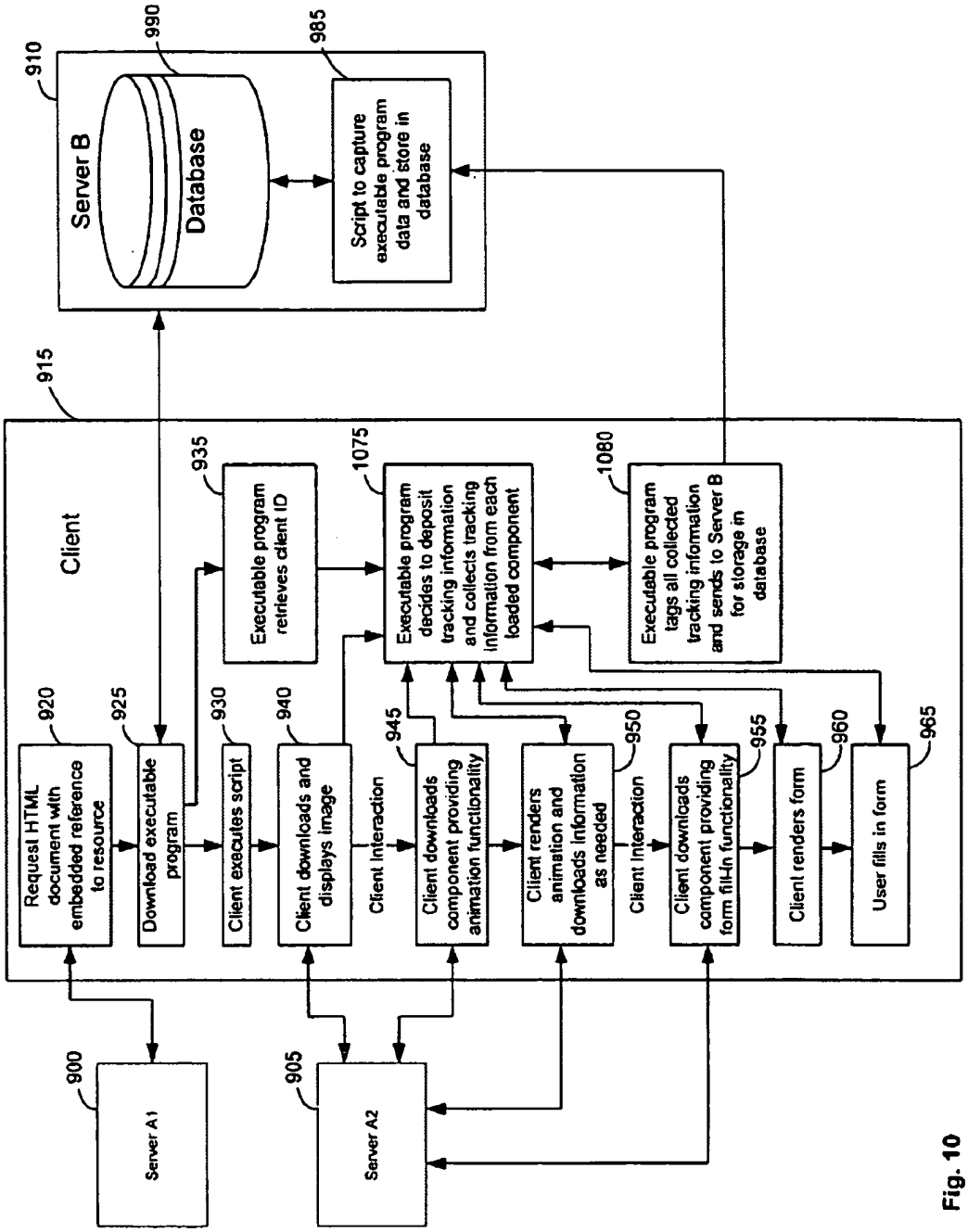

910
990
985
Server B
Database
Script to capture executable program data and store in database
915
Client
935
Executable program retrieves client ID
1075
Executable program decides to deposit tracking information and collects tracking information from each loaded component
1080
Executable program tags all collected tracking information and sends to Server B for storage in database
920
Request HTML document with embedded reference to resource
925
Download executable program
930
Client executes script
940
Client downloads and displays image
945
Client interaction
Client downloads component providing animation functionality
950
Client renders animation and downloads information as needed
Client interaction
955
Client downloads component providing form fill-in functionality
960
Client renders form
965
User fills in form
900
Server A1
905
Server A2

Fig. 10

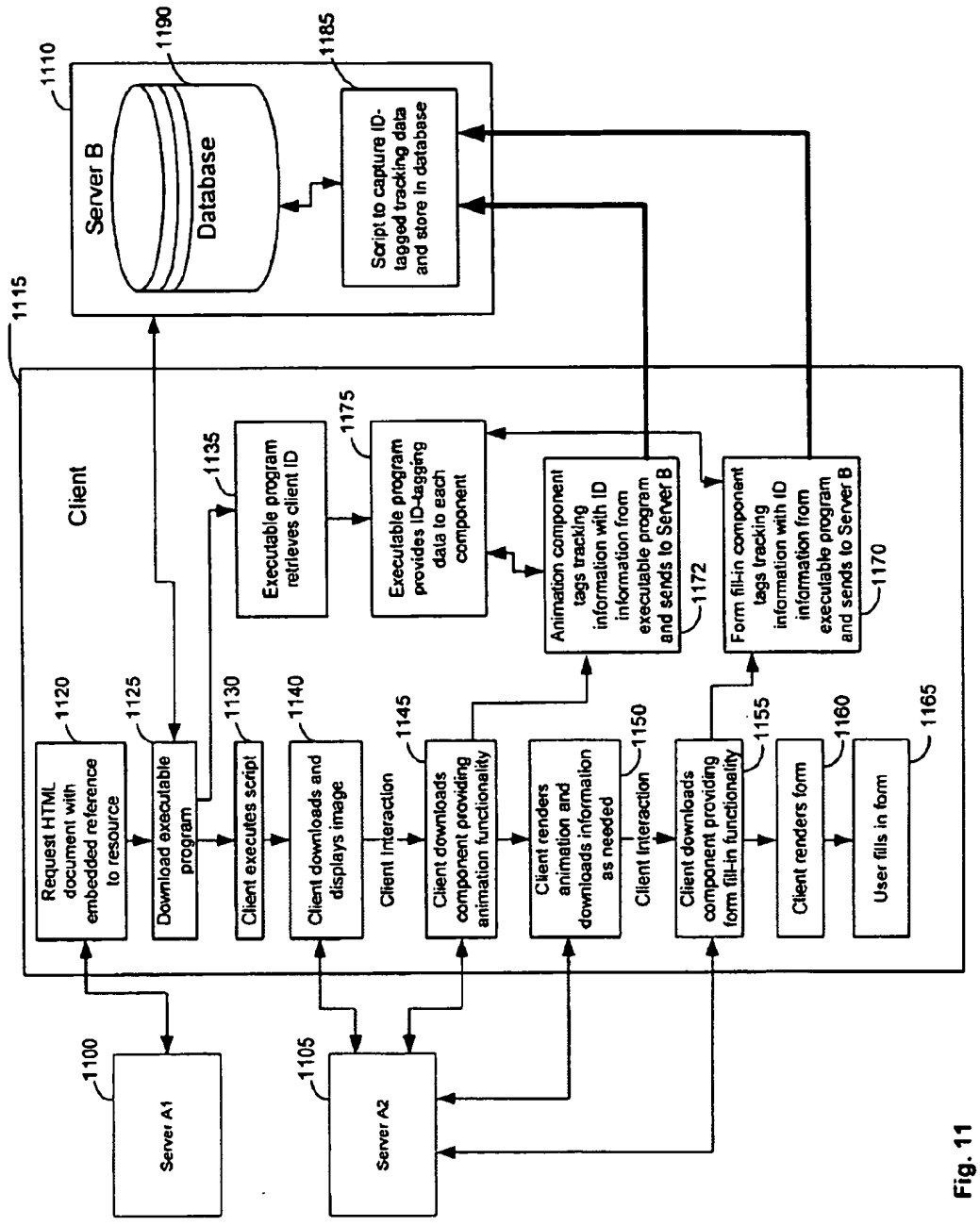

1110
1190
1185
Server B
Database
Script to capture ID-tagged tracking data and store in database
1115
Client
1135
Executable program retrieves client ID
1175
Executable program provides ID-tagging data to each component
Animation component tags tracking information with ID information from executable program and sends to Server B
1172
Form fill-in component tags tracking information with ID information from executable program and sends to Server B
1170
1120
Request HTML document with embedded reference to resource
1125
Download executable program
1130
Client executes script
1140
Client downloads and displays image
Client Interaction
1145
Client downloads component providing animation functionality
Client renders animation and downloads information as needed
1150
Client Interaction
Client downloads component providing form fill-in functionality
1155
Client renders form
1160
User fills in form
1165
1100
Server A1
1105
Server A2

Fig. 11

METHOD AND APPARATUS FOR TRACKING CLIENT INTERACTION WITH A NETWORK RESOURCE AND CREATING CLIENT PROFILES AND RESOURCE DATABASE

RELATED APPLICATIONS

This application is a continuation-in-part of application Ser. No. 09/120,376, titled "METHOD AND APPARATUS FOR TRACKING CLIENT INTERACTION WITH A NETWORK RESOURCE AND CREATING CLIENT PROFILES AND RESOURCE DATABASE", filed Jul. 21, 1998, now U.S. Pat No. 6,138,155, which is a continuation of application Ser. No. 08/821,534, titled "METHOD AND APPARATUS FOR TRACKING CLIENT INTERACTION WITH A NETWORK RESOURCE AND CREATING CLIENT PROFILES AND RESOURCE DATABASE", filed Mar. 21, 1997, issued Aug. 18, 1998, as U.S. Pat. No. 5,796,952, the disclosures of which are herein incorporated by reference.

COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

FIELD OF THE INVENTION

The present invention relates to a method and apparatus for monitoring client use of and interaction with a resource downloaded from a server on a computer network, for storing monitored data, for creating a database including profiles indexed by user and/or resource identity, and for generating customized resources based upon client profiles.

BACKGROUND OF THE INVENTION

The development of software packages designed to permit simplified graphical user interface (GUI)-based access to the wealth of electronic information available over the Internet and the World Wide Web has lead to a dramatic increase in the amount of information that is currently available over public computer networks. Unlike the highly controlled atmosphere of a private computer network, however, it is difficult to monitor user interaction with network resources on public networks. As a result, it is difficult for individual servers on a public network to determine how long individual users have interacted with their resources, or how much information has been downloaded. It is equally difficult for individual servers to target specialized information to a particular audience or to learn the identity of individual users on a public network.

The techniques utilized in many private networks for monitoring client use and interaction do not lend themselves to public networks. For example, user access to a server in private networks is generally obtained through the use of a unique identification number provided by the server. Details of individual user interaction with the network are closely monitored by server-resident processes, and historic databases are automatically generated and continually updated to track the nature and amount of information accessed by individual users, as well as their connection time. This information is generally used, for example, to maintain a subscriber-indexed billing database.

In a public computer network, however, use of server-resident monitoring techniques may be severely limited. In some public networks, subscribers are given unlimited access, via a service provider, to a virtually unlimited number of servers, and no permanent connection is usually made between these servers and a client machine. The nature and amount of information downloaded by individual users is not easily monitored for each client machine and only limited information concerning individual user interaction with the network may generally be captured by a server (i.e., so-called network ID and client ID).

Due largely to the lack of advanced monitoring techniques available to individual servers on a public network, the same information is generally served out to all clients on a completely untargeted basis. In other words, the same information is generally downloaded to all users that access a particular resource on a server, irrespective of individual user interests. There is therefore a need to provide servers on a public network with the ability to automatically monitor use of and interaction with resources downloaded by users so as to facilitate the targeted serving of information.

While various methods are known for obtaining information concerning user preferences, no such methods are automatic. For instance, one application, known as a "customizable home page", permits users, upon the request of a server, to make certain choices. When a user who has done so contacts that server at a later date, the server assembles information for downloading to the user in accordance with the previously-selected choices. More specifically, the user visits a so-called "Web page" of a particular server where he or she is asked to fill in a blank form by selecting various preferences, such as links to favorite Web sites, interests in entertainment, sports, and the like. The user then submits this information to the server by clicking the so-called "submit" button of the fill-in form, which causes the client to transmit the information to the server. The server returns a Web page with a response header which creates, or "sets" an ID field located in a file on the client computer (this file is known as the "client ID" or "cookie") to include information about the user's preferences. When the user later returns to a specified Uniform Resource Locator, or "URL", on the same server, the "client ID" or "cookie" with the previously-set preference information is transmitted in the HTTP request header to the server, which can then return a Web page that is assembled according to the user-specific information. This application is disclosed, for example, in A. Gundavaram, CGI Programming on the World Wide Web, O'Reilly Press, 1996.

While the "customizable home page" facilitates the serving of information on a limited targeted basis, it does not provide for the automatic determination of user interests, and inconveniences the user by requesting that he or she specify various preferences. Moreover, use of a customizable home page is limited to individual Web sites and can not be "spread out" over multiple resources on different servers. In other words, while a customizable home page may be of use with respect to the particular resources located on a single server, it does not serve any purpose for other servers on a public network. A variation of this technique is used by some servers to download executable programs. For instance, one such application disclosed by G. Cornell and C. S. Horstmann, in Core Java, The SunSoft Press, 1996, involves the generation of "order forms" on client computers. In this application, the client machine loads a Web page from a server which has an embedded link to an executable program that downloads to and executes on the client machine. Upon execution in the client machine, the program

contacts the server and retrieves a list of goods and associated prices. The program allows the user to order various goods and requires the user to fill out a form for billing purposes. The user "clicks" on the submit button of the fill-in form to transmit the information to the server. Like the customizable home page, this method of user-specific data acquisition requires the active participation of the user, and does not provide for the automatic determination of user preferences and interests.

In addition to the inability to serve out information on a targeted basis, which is of enormous concern from a marketing standpoint, the limited monitoring capabilities available to individual servers makes it difficult for servers and administrators to determine how long users have viewed their resources and how much information has been downloaded by individual users so as to be able to bill client use and interaction with network resources and to analyze the value and effectiveness of such resources. As a result, much of the information provided by a server over a public network is the same for all clients. In addition, while it is currently possible to track a user's links within the same resource, there is no standard way to track user's links across multiple resources on different servers. For example, a common occurrence in public networks is when a user is viewing a first resource and "clicks on" a link to a second resource located on a different server. In such instances, the second resource is downloaded and the first resource is either discarded or held in background. However, there is generally no uniform way in which to monitor such occurrences. In addition, while it is currently possible to track the number of times a particular resource has been accessed, it has generally not been possible to track the length of time a particular resource has been viewed by a particular user. There is also a great deal of other valuable information concerning user interaction with a resource which would be useful to administrators, advertisers, marketing professionals and the like, but which can not be conveniently collected using current monitoring techniques.

For example, one of the largest public networks, the "Internet", has become an extremely popular advertising tool. Many companies have their own Internet "Web sites" and have also purchased advertising space within more popular Web sites of other companies. For instance, many advertisers purchase so-called "advertising banner" (or "ad banner") space within the Web page of a popular site, thereby allowing consumers to "click-through" (i.e., specify a link) to the Web site of the advertiser. In many cases, the use of an ad banner substantially increases the advertiser's exposure. Using the limited monitoring techniques available to Internet servers, however, it is difficult to determine the effectiveness of individual Web sites and ad banners. For instance, known monitoring techniques are generally limited to determining the number of times a Web page was downloaded. Similar techniques are used to determine the number of times an ad banner (which is embedded inside a Web page) has been displayed, and how many times the banner was "clicked" on to visit the Web site of the advertiser.

Generally, an ad banner is embedded inside a Web page located on a first server through the use of the known HTML<IMG> tag. When a client machine passes a TCP/IP request for the Web page to the first server, the Web page is downloaded to the client, including the ad banner embedded using the <IMG> tag. The <IMG> tag is used to reference a resource (i.e., the "ad banner") stored on the same or a different server which captures the user's ID (via the HTTP request header) and dynamically returns an ad related image to the client for display within the Web page. At the same time, a counter representing the number of times the specific ad has been displayed is incremented. The ad banner itself may have an embedded address referring to yet another Web resource. In such an instance, if the user "clicks" on the ad banner, the client may load a resource on the second server which once again captures the user's ID and forwards the user to a Web resource which is appropriate for the displayed ad (for example, a page on the advertiser's Web site). At the same time, a counter representing the number of times the specific ad was clicked on to go to the advertiser's Web site is incremented.

While Web sites and ad banners have, in some cases, been valuable marketing tools, the limited monitoring capabilities available to servers on networks in which no permanent connection is made between a server and a client (such as the Internet) has prevented these marketing tools from being used to their fill potential. Since HTTP or Web servers cannot automatically determine the amount of time and the frequency at which particular users interact with their resources, Web site administrators and advertisers cannot accurately determine the effectiveness of their resources. Since servers cannot automatically monitor user interaction and automatically obtain user preferences and interests, servers cannot assemble and serve resources targeted to individual user interests.

BRIEF SUMMARY OF THE INVENTION

In view of the foregoing shortcomings of the prior art, an object of the present invention is to provide a method for tracking the use and interaction of a user with a resource downloaded from a server on a network by use of a tracking program embedded in the resource and executable by a client. Another object of the present invention is to transmit the tracking information from a client to another computer connected to the network for storage and analysis.

Still another object of the present invention is to create a database of server resources including, but not limited to, the number of times a resource has been displayed by clients, the amount of time displayed, and the type and amount of information that was displayed or transferred. This information could be used by network administrators or servers to analyze the effectiveness of the resources made available on their network servers.

Yet another object of the present invention is to provide means for creating a database of user profiles for use by advertisers and/or marketers to determine the effectiveness and value of network-based advertisements and/or marketing resources.

Still yet another object of the present invention is to provide means for creating a database of user profiles containing details of individual user interaction with and use of network resources including, for example, Network IDs (known as "IP address") and client IDs (known as "cookies") that have accessed particular resources, the amount of time spent by users interacting with and/or using particular resources, and details of choices created by individual users within a particular resource.

It is still yet another object of the present invention to provide means for assembling a resource, such as a Web page or a highly targeted ad banner, in accordance with a historic user profile.

In order to achieve the above-described and other objects and advantages, a tracking program is embedded in a file which is downloaded from a server to a client. The tracking program need not originate from the same server that sent the file, and may be obtained, for example, via an embedded

URL that points to a different server. The tracking program may be part of a larger program that performs other operations (such as displaying animations, playing sounds, etc.). The tracking program is downloaded from a server and runs on the client to monitor various indicia, such as elapsed time, mouse events, keyboard events, and the like, in order to track the user's interaction with and use of the file or to monitor choices (such as selections or links to other resources or files) made by the user while within the file. The tracking program may also monitor the amount of data downloaded by the client. Operation of the tracking program commences after the program is downloaded and any required initialization occurs.

After monitoring the user's interaction with and use of the file downloaded from the server, the tracking program then automatically sends the information acquired from the client back to a server for storage and analysis. The information may be sent before or as the client exits the file, or may be sent in response to a predetermined user action. The information preferably includes any available client or network IDs.

The acquired information is preferably stored on a server and used to build historical profiles of individual users, to serve out highly targeted information based upon user profiles, as well as to extract information about how much data was downloaded by a respective client, and how long or how often specific files were displayed or in use by the client.

Preferably, the tracking program is implemented in a network based upon the client/server model, and may be implemented in a public network such as the Internet or World Wide Web. The tracking program may monitor use of and interaction with any of the resources downloaded from a server, including an executable program, a database file, an interactive game, a multimedia application, and the like. In the case of the Internet, for example, the tracked resource may, for example, be a file such as a Web page or part of a Web page (such as an ad banner).

In one embodiment of the present invention, the tracking program is embedded in an HTML document (such as a Web site, a Web page, or part of a Web page—e.g. an "ad banner"). A TCP/IP connection is used by a client to pass a request for the HTML document. The HTML document is stored in a server running an HTTP service and contains text and one or more first embedded URLs for pointing to one or more graphical images located on a server, the images being embedded inside the HTML document using an HTML<IMG> tag to specify the source URL for an image. The HTML document also contains a second embedded URL for pointing to a first executable program that runs on a server, the first executable program being embedded inside the HTML document using an HTML<IMG> tag to specify the source URL for the program. A second executable program is also embedded in the HTML document by using a third URL for pointing to the second executable program. Unlike the first executable program, the second executable program is downloaded and runs on the client. The second executable program is embedded using the proper HTML tag to indicate that it is a program that is executable on the client.

After the HTML document is downloaded to the client, the graphical images are fetched using a TCP/IP connection to server resources specified by the one or more first URLs. In attempting to fetch the resource associated with the first executable program, the client causes the program to run on the server specified by the second URL. Upon execution of the first executable program, the server captures identifying indicia from the client, such as any network or client IDs resident in the HTTP request header sent by the client. The server stores this information in a client profile database.

The client also fetches the second executable program, which is the tracking program. The tracking program downloads to the client, and, after performing any required initialization, determines the current time. The tracking program also determines the current time upon the performance of a predetermined operation on the client computer by a user, such as leaving the HTML document. After calculating the amount of time the user interacted with and displayed the HTML document, i.e., by determining the difference in time values, the tracking program uploads the calculated value to the server for storage in the user profile database.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a diagram of a computer network in which the present invention may be implemented;

FIG. **2** is a block diagram of a client computer which is used in connection with various preferred embodiments of the present invention;

FIG. **3** is a flowchart diagram of a first embodiment of the present invention, which is a method for monitoring the amount of time a Web page is displayed on a client computer;

FIG. **4** is a flowchart diagram of a second embodiment of the present invention, which is a method for monitoring the amount of time a Web page is displayed on a client computer;

FIG. **5** is a flowchart diagram of a third embodiment of the present invention;

FIG. **6** is a flowchart diagram of a fourth embodiment of the present invention;

FIG. **7** is a flowchart diagram of a fifth embodiment of the present invention;

FIG. **8** is a flowchart diagram of a sixth embodiment of the present invention;

FIG. **9** is a flowchart diagram of a seventh embodiment of the present invention;

FIG. **10** is a flowchart diagram of a eighth embodiment of the present invention; and

FIG. **11** is a flowchart diagram of a ninth embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The teachings of the present invention are applicable to many different types of computer networks and may also be used, for instance, in conjunction with direct on-line connections to databases. As will be appreciated by those of ordinary skill in the art, while the following discussion sets forth various preferred implementations of the method and system of the present invention, these implementations are not intended to be restrictive of the appended claims, nor are they intended to imply that the claimed invention has limited applicability to one type of computer network. In this regard, the teachings of the present invention are equally applicable for use in local area networks of all types, wide area networks, private networks, on-line subscription services, on-line database services, private networks, and public networks including the Internet and the World Wide Web. While the principles underlying the Internet and the World

Wide Web are described in some detail hereinbelow in connection with various aspects of the present invention, this discussion is provided for descriptive purposes only and is not intended to imply any limiting aspects to the broadly claimed methods and systems of the present invention.

The present invention, although equally applicable to public and private computer networks, is particularly useful for performing monitoring functions in connection with public networks which could not heretofore be performed. For this reason, implementation of the present invention will be discussed in detail in connection with the Internet and the World Wide Web. This discussion is equally applicable to any network based upon the client/server model.

Accordingly, as will be appreciated by those of ordinary skill in the art, as used herein, the term "client" refers to a client computer (or machine) on a network, or to a process, such as a Web browser, which runs on a client computer in order to facilitate network connectivity and communications. Thus, for example, a "client machine" can store and one or more "client processes." The term "user" is used to broadly refer to one or more persons that use a particular client machine.

FIG. 1 illustrates a known computer network based on the client-server model, such as the Internet. The network comprises one or more "servers" **10** which are accessible by "clients" **12**, such as personal computers, which, in the case of the Internet, is provided through a private access provider **14** (such as Digital Telemedia in New York City) or an on-line service provider **16** (such as America On-Line, Prodigy, CompuServe, the Microsoft Network, and the like). Each of the clients **12** may run a "Web browser", which is a known software tool used to access the Web via a connection obtained through an Internet access provider. The servers allow access to various network resources. In the Internet, for example, a Web server **10** allows access to so-called "Web sites" which comprise resources in various different formats. A location of a resource on a server is identified by a so-called Uniform Resource Locator, or URL.

The "World Wide Web" ("Web") is that collection of servers on the Internet that utilize the Hypertext Transfer Protocol (HTTP). HTTP is a known application protocol that provides users access to resources (which can be information in different formats such as text, graphics, images, sound, video, Hypertext Markup Language-"HTML" etc., as well as programs). HTML is a standard page description language which provides basic document formatting and allows the developer to specify "links" to other servers and files. Links are specified via a Uniform Resource Locator or "URL". Upon specification of a link, the client makes a TCP/IP request to the server and receives information that was specified in that URL (for example another "Web page" that was formatted according to HTML) in return. The information returned may be generated in whole or in part by a program that executes on the server. Such programs are typically known as CGI (Common-Gateway-Interface) scripts and can be written using known programming languages or methods that the server supports, such as PERL or C++. A typical Web page is an HTML document with text, "links" that a user may activate (e.g. "click on"), as well as embedded URLs pointing to resources (such as images, video or sound) that the client must fetch to fully render the Web Page in a browser. These resources may not be located on the same server that the HTML document was sent from. Furthermore, HTTP allows for the transmission of certain information from the client to a server. This information can be embedded within the URL, can be contained in the HTTP header fields, or can be posted directly to the server using known HTTP methods.

FIG. 2 is a block diagram of a representative "client" computer. The same or similar computer can also be used for each of the servers. The system unit **21** includes a system bus **31** to which various components are coupled and by which communication between the various components is accomplished. The microprocessor **32** is connected to the system bus **31** and is supported by a read only memory (ROM) **33** and random access memory (RAM) **34**. The ROM **33** contains, among other code, the basic input-output system (BIOS) which controls basic hardware operations such as the interaction and the disk drives and the keyboard. The RAM **34** is the main memory into which the operating system **60** and application programs, such as a Web browser **62**, are loaded and cached **63**. The memory management chip **35** is connected to the system bus **31** and controls direct memory access operations, including passing data between the RAM **34** and the hard disk drive **36** and the floppy disk drive **37**. The CD ROM **42**, also coupled to the system bus, **31**, is used to store a large amount of data, e.g., multimedia programs or large databases.

Also connected to the system bus **31** are various I/O controllers: the keyboard controller **38**, the mouse controller **39**, the video controller **40**, and the audio controller **41**. The keyboard controller **38** provides the hardware interface for the keyboard **22**, the controller **39** provides the hardware interface for the mouse (or other hand-operated input implement) **23**, the video controller **40** provides the hardware interface for the display **24**, and the audio controller **41** is the hardware interface for the multimedia speakers **25***a* and **25***b*. A modem **50** (or network card) enables communication over a network **56** to other computers over the computer network. The operating system **60** of the computer may be Macintosh OS, OS/2, AIX, BE OS or any other known operating system, and each client computer is sometimes referred to as a "client machine", a client "computer", or simply as a "client."

As noted above, the Internet includes a public network using the Internet Protocol (TCP/IP) and includes servers **10** which are accessible by clients **12**. When a Web browser **62** is used to access a file on a server **10**, the server **10** may send information including graphics, instruction sets, sound and video files in addition to HTML documents (Web pages) to the requesting client.

In accordance with the present invention, a tracking program is embedded in a resource, such as an HTML document which is sent from a server to a client based on a TCP/IP request. The tracking program may originate on a different server than the resource, in which case it may be obtained by the client through a TCP/IP request to the other server. The tracking program executes on a client machine, and is stored, for example, in RAM. The tracking program may monitor various indicia, such as time, mouse events, keyboard events, and the like, in order to track a user's interaction with the Web page. Thus, the tracking program may simply monitor the amount of time the user spends interacting with the Web page, or may monitor details of choices (such as links) made by individual users within a particular Web page.

In some cases, clients will "cache" a resource obtained over the network (or temporarily store a copy of the resource on the user's computer), and may use the cached copy of the resource instead of obtaining it over the Internet when the resource is needed at a later time (for example, in order to completely render a Web page). In such cases, neither the basic operations nor functions of the tracking program nor the transmission of tracked information to a server, differ from the cases where cached copies were not used.

In one embodiment of the present invention, a tracking program is embedded in an HTML of a Web page and downloaded by a client. The tracking program may monitor operation of a peripheral input device connected to the client machine, such as a keyboard or mouse, keep a record of which choices, if any, are made by a user, and may monitor the length of time the user has displayed the Web page in addition to the time spent interacting with a particular part of it. While in the preferred embodiment, the tracking program is embedded in an HTML document, those skilled in the art will recognize that other mechanisms are possible for embedding the tracking program in the client hardware, and the patent is not limited to implementation as an executable program embedded in an HTML document. For example, the tracking program may be downloaded and installed in a client process, as would be the case for a so-called "plug-in" or "helper" application. Alternatively, the tracking program can be built into a client application or client process such that it need not be separately downloaded and installed. In addition, the teachings of the present invention are not limited to use on the Internet or the World Wide Web. For instance, the tracking program of the present invention may be utilized on a so-called "Intranet".

As noted above, a client process, such as a Web browser running on the client machine, uses a TCP/IP connection to pass a request to a Web server running an HTTP service (or "daemon" under the UNIX operating system). The HTTP service then responds to the request, typically by sending a Web page formatted in the Hypertext Markup Language, or HTML, to the browser. The browser displays the Web page using local resources (e.g., fonts and colors). Unless the tracking program is already resident in the client, it is embedded in the Web page and downloaded to the client along with the Web page. The tracking program is executed after any required initialization has occurred. The tracking program may monitor the length of time the user remains in the Web page, or any one or more portions thereof, and may track some or all mouse and keyboard events to provide meaningful data to the server concerning the user's interaction with the Web page.

In its simplest form, the tracking program is a timer program linked to an HTML document and is downloaded and executed on a client when the HTML document is served to the client in response to a client TCP/IP request. During or after the client formats and displays the Web page specified by the HTML document, the tracking program begins a software timer to monitor the amount of time the Web page is displayed on the client computer.

When the user leaves the Web page (for example, by exiting the Web page or "clicking" on a link to another resource on the same or another server), the tracking program sends the monitored time to another computer on the Internet for storage and analysis.

As illustrated, for example, in FIG. 3, the client issues a TCP/IP request for a Web page located on a Server A (S301). After a handshaking period, the Server A begins to send the HTML formatted document, which contains an embedded URL referencing the tracking program. The client additionally issues a TCP/IP request to the Server B referenced by the embedded URL in order to obtain the tracking program (S302). The client also makes any other TCP/IP requests (S303) to obtain any other resources (such as images, video or sound) needed in order to fully render the Web Page (S304). Each of such resources are typically referenced by individual URLs embedded in the HTML document. These requests need not occur in any specific order and may reference resources located on any server. In addition, the information requested may be received in any order. When the tracking program has been obtained, the client process (i.e., the Web browser) saves the tracking program to RAM (S305). After any necessary initialization, the tracking program initiates a software timer to monitor the amount of time the Web page is displayed (S306). When the client leaves the Web page (S307), the tracking program calculates the amount of time the user has interacted with and displayed the Web page and sends this information to a server. Other available client information, such as the network ID and client ID, or so-called "Cookie" of the client, is also sent to the server (S308). If desired, other information concerning the client computer may be automatically acquired and sent to the server, such as the type of hardware in the client computer and various resources that are resident on the client computer.

Due to the technical limitations imposed by the Internet, the JAVA programming language was applied to the Internet in 1995 by programmers at Sun Microsystems, Inc. of Mountain View, Calif. For example, some of the fundamental technology issues facing network programmers and engineers are portability, bandwidth and security. Portability allows the same executable code to run across multiple operating systems. Bandwidth specifies the amount of information that can transfer across the network at any time. For instance, high-speed lines categorized as T1 through T3 can transmit data at 1.544 through 45 megabits per second, ISDN lines can transmit data at rates of 64 through 128 kilobits per second, and standard phone lines, over which most users transmit data, currently transmit using modems at approximately 28.8 kilobits per second. In the case of a 640×480 pixel window on a computer display that is capable of displaying images in 256 colors (which requires one byte per pixel), in order to display the window's contents requires 307,200 bytes of data. To create an animation, programs typically display 15 to 30 different images per second. Given a 640×480 window, 15 to 30 frames per second would require 4,608,000 to 9,216,000 bytes per second. Because many users are currently browsing the Web using 28.8 kilobit (or slower) modems, there is simply not enough bandwidth to download animation screens. As a result, many Web sites today resemble magazines whose images are for the most part static (unchanging). However, to satisfy an audience that spends many hours in front of dynamic television images, Internet programmers and engineers must provide a way to animate Web sites. One solution is to download programs written in the JAVA programming language that implement the animation.

Animation is only one example of the use of JAVA. Using JAVA, programmers can create stand alone programs similar to those that programmers can develop using C++, and can also create so-called "applets" that run within a Web browser. To address security issues, JAVA developers ensured that a programmer could not develop a computer virus using a JAVA applet and that an applet could not arbitrarily transfer information concerning a user's system (such as a file on the user's system) back to the server. Thus, JAVA applets have limited operations. For example, a JAVA applet generally cannot currently read or write files on the user's system. In this way, an applet cannot store a virus on a user's disk or arbitrarily read information stored on a user's disk. In addition, for other security and stability reasons, JAVA developers eliminated or changed many features of the C and C++ programming languages, such as pointers, with which advanced programmers could bypass JAVA's security mechanisms.

JAVA applets run within a "JAVA-enabled client", such as Netscape Navigator version 2.0 (Windows 95 or Windows

NT versions only) or later, or Microsoft's Internet Explorer version 3.0, or later. In addition, since most users browse with personal computers running Windows, Macintosh, UNIX-based systems, and the like, the JAVA developers designed JAVA to be portable, or "platform-independent". Thus, the same JAVA applets can be downloaded and run in any JAVA-enabled client process, irrespective of the platform type.

JAVA applets can be used by developers to create sophisticated, fully interactive multimedia Web pages and Web sites executable on any JAVA-enabled client. Representative JAVA applets are disclosed, for example, by O. Davis, T. McGinn, and A. Bhatani, in Instant Java Applets, Ziff-Davis Press, 1996.

Since JAVA provides the ability to download complex programming instructions in the form of applets that are executable by a JAVA-enabled Web browser, the tracking program of the present invention may be implemented in the JAVA programming language. As will be readily appreciated by those of ordinary skill in the art, however, the teachings of the present invention are not limited to JAVA applets or to the JAVA programming language whatsoever. In connection with the Internet, for example, the present invention may also be implemented in a so-called "Active-X" environment, in which the tracking program is written as an Active-X component.

As will be further appreciated by those of ordinary skill in the art, security restrictions may, in some cases, prevent one from having direct access to information stored on a client's hard disk, such as client IDs. In such cases, other means may be used to obtain this information. For example, when a Web browser makes a request for information from a server it typically includes certain information about the client in the "HTTP request header." The server receiving the request can obtain and store this information using known means implemented, for example, in a so-called "CGI script" executable on the server. Therefore, one way of obtaining client identifying indicia is to embed a request in the HTML file for another resource on a server that will obtain and store the indicia. This resource may be a program (such as a CGI script) that captures relevant information and stores it. This information can then be combined with information monitored by the tracking program to provide a more detailed knowledge base. This embedded request may be in addition to the embedded tracking program. Representative CGI scripts capable of capturing client identifying indicia are disclosed by A. Gundavaram, in CGI Programming on the World Wide Web, O'Reilly Press, 1996.

In order to store client-identifying indicia, such as a user's network ID (IP) and client ID numbers (cookies) and associated tracking information, a database is set up on a server. This may be done in any known manner, such as by using a commercially-available database program designed, for example, for the high-speed processing of large databases. In the case of the tracking program described above, the information stored in the server database may include the network ID, client ID, the associated link (the URL of the Web page), the amount of time the user spent interacting with the Web page, and any selections or choices made by the user while interacting with the Web page. Thus, the above-described tracking program permits Web site administrators and Internet advertisers, for example, to determine not only the number of user visits or hits made to a particular Web page, but also permits the accurate determination of the length of time users have displayed and/or interacted with their Web page. This is invaluable information to Internet advertisers, among others, and permits advertisers to make informed decisions as to the effectiveness and value of particular Web pages and/or ad banners.

A more particular embodiment of this aspect of the invention is illustrated in FIG. 4. A Web page (or HTML document) is requested by the client from a first server A, using TCP/IP and HTTP protocols (S401). This HTML document contains text, as well as embedded URLs that point to graphical images (e.g. GIF format image files) also located on the first server A. The images, in general, may be located on any HTTP server on the Internet. These images are embedded inside the Web page using the known HTML<IMG> tag, which allows one to specify the source URL for an image, as well as additional information such as size and other layout parameters. These images will then be fetched by the client using TCP/IP and HTTP protocols from Server A (S402) and rendered on the browser (S405). The Web page (or other Web or HTML document) additionally includes embedded URLs which point to two resources that reside on a second server "B". One of the resources is an executable program, which executes on Server B, and is a CGI script. This resource is also embedded inside the Web page using the <IMG> tag. Thus, in attempting to render the Web page, the client will automatically fetch this resource (S403), which forces execution of the CGI script on the second Server B and the return of information output from the script to the client. In this case, the information returned to the client is formatted as an GIF image type which is extremely small as well as completely transparent (S403B). When the CGI script executes, it may collect information from the HTTP request header such as browser type, network ID (IP address), and if set, client ID ("cookie"), as well as any additional available information such as time of execution and the URL of the Web page, and store it in a database—for example using SQL (S403A, S404). In step S403B, the CGI script returns information to the client, which includes a response header which indicates (among other information), that the return type is an image, that this resource should not be cached by the client, and if no client ID is set and the client supports it, that a client ID is to be set to a value generated by the script.

In addition, the CGI script may monitor the number of times the Web page has been accessed in general. On the other hand, another CGI script located on the same or another server may be used for this purpose. This process may be carried out by simply incrementing a counter each time the resource is accessed, or may be conducted at any other time by merely counting the number of entries made in a stored record of requests made for the resource.

The other resource located on Server B is a JAVA applet, the tracking program. This resource can also be located on any other server, and is embedded in the Web page using the known HTML<APPLET> tag, which allows one to specify the source URL (through the CODE and CODEBASE parameters) as well as additional size, layout and initialization parameters. The client, in attempting to render the Web page, will automatically fetch the applet by making a request to Server B using the TCP/IP and HTTP protocols (S406). Soon after it has received the JAVA code for the tracking program, it will first execute the INIT (initialization) method of the applet (S407) and then the START method. The START method will make note of the current time using standard JAVA methods (S408). The STOP method of the applet which is executed, for example, when the user leaves the Web page (S409), will compute the difference between the current time and the time noted during execution of the START method. This difference, which is the time between execution of the STOP and execution of the START

methods, is sent to the Server B for storage and analysis (S410). The information can be sent using standard JAVA network methods, such as opening a URL connection to a second CGI script on Server B (or any other server) designed to capture the tracked information (S410A). This second CGI script can then obtain any information tracked and transmitted by the applet as well as any available information in the HTTP request header. This information can be stored in a database on Server B or elsewhere. If necessary, the information stored by both scripts may be combined into one or more complete databases. As will be understood by those of ordinary skill in the art, acquisition of information by the server need not be conducted using CGI scripts. For instance, this information may be acquired by any other server-resident process designed for this purpose, or may be uploaded by the tracking program or other client-resident process, such as by a direct connection to a resource located on a server (i.e., a database), or by using any other known process.

The database thus constructed can be indexed by resource identity and may contain information about users who have visited the Web page, such as their network and client IDs, how often they visited the Web page, how long the Web page was displayed, and so on. Additionally, if the above-mentioned tracking mechanism is implemented across various Web pages in a particular Web site, the database thus constructed may contain similar information about the different Web pages in the Web site. Similarly, the information acquired by the tracking program may be combined with a process for monitoring the number of times the Web resource has been accessed. An analysis of the data on a user-indexed basis would facilitate the determination of individual user interests and the like. On the other hand, analysis of the data on a resource-indexed basis would allow the determination of, for example, which Web pages are viewed the longest and/or most often either by users in general, or by specific users. Thus, it would be possible to determine if there were different types of users that preferred different sections of the Web site (because, for example, they spent more time browsing different sections of the Web site). Additionally, if the above-mentioned tracking program is attached to an ad banner that is embedded in multiple Web pages across different Web sites (as is typically the case with ad banners), the database thus constructed may contain information about how often and for how long the different pages that contained the ad banner were displayed, as well as more specific information about users that visited those pages. With this information, advertisers could determine the accuracy of data supplied to them by Web site administrators about the number of times their ad banner was displayed, as well as learn how long the Web page containing the ad banner was displayed—a number that would be of great use in determining the effectiveness of their advertising.

In another embodiment, the software timer of the tracking program may be initiated or stopped when the user incurs a keyboard or mouse event, such as by "clicking" on a specified area of an ad banner. This is illustrated in the flowchart shown in FIG. 5. Operation of the system in this embodiment is similar to that shown in FIG. 3. Thus, the client first issues a TCP/IP request (S501). After a handshaking period, a first Server A begins to send an HTML formatted document, which contains an embedded URL referencing the tracking program. The client additionally issues a TCP/IP request to a second Server B referenced by the embedded URL in order to obtain the tracking program (S502B). The client also makes any other TCP/IP requests to obtain any other resources (such as images, video or sound) needed in order to fully render the Web Page (S502A). Each of such resources are typically referenced by individual URLs embedded in the HTML document. These requests need not occur in any specific order, and the information requested may be received in any order. When the tracking program has been obtained, the client process (i.e., the Web browser) saves the tracking program to RAM (S503B). In this case, the tracking program commences a software timer upon the detection of a predetermined user action (S504). When the user performs another predetermined action (S505), the tracking program calculates the amount of time between the predetermined user actions, and sends this information, along with other available client information, to the server (S506).

Thus, for instance, the software timer of the tracking program may be used in monitor the amount of time a user spends interacting with a portion of a Web page. For example, if the Web page is provided with an interactive resource such as a game or an information resource activated by clicking on a particular button, the tracking program may determine how long a user has interacted with such a selection. In the case of a Web page provided with an ad banner, the tracking program can be designed to monitor the amount of time a user has interacted with the ad banner.

The tracking program may be used not only to monitor the time spent by a user in a Web page or an ad banner, but may also be used to create a more complex "historical" user profile to permit the server to assemble a Web page or target an ad banner based upon the diverse interests of respective users.

For example, when a user is exposed to an ad banner having information targeted to their particular interests, the user is more likely to interact with that ad banner for a longer period of time and on a more frequent basis, thereby increasing the value of that ad banner. In accordance with the present invention, in order to learn the particular interests of respective users, an ad banner may include specific information permitting the user to interact in different ways with the banner. The ad banner may have pull-down menu options, clickable buttons or "hot-spots", keyboard input, or any number of input mechanisms, whose selection or action upon in a designated manner causes corresponding events to take place in the ad banner such as the generation or synthesis of sounds, the display of images, video, or graphic animations, or the presentation of different types of information to the user, perhaps with additional choices. Such information may, for example, include links to interactive games, links to entertainment information, sports-related games and/or trivia, and the like, or information concerning particular goods and services, or means by which to order or purchase specific goods and services. The more choices that are made available, the more information that can be acquired concerning the user's particular interests. Of course, an unlimited number of possibilities are available, depending upon the application, and an exhaustive listing of such possibilities cannot be provided herein.

In this case, the tracking program is downloaded, as described above, with the HTML document in response to a TCP/IP client request. As above, the tracking program may monitor the amount of time the user spends displaying both the Web page and the ad banner embedded in the Web page as a whole, but also monitors the user's interaction with the Web page and the ad banner, such as by monitoring each of the choices made by the user within the Web page and ad banner. Thus, for example, if an interactive sports-related game is included in the Web page, the tracking program will

determine if a user has played the game, what his or her score was, how long they played the game, and any other possible information. If a choice of different games, each directed to a different interest, are made available to users within the same ad banner, it is possible to determine what is of most interest to the user by the selection of the game. In addition, the ad banner may be provided with multiple links to other, diverse Web sites, such as Web sites relating to sports, entertainment, general information, technology, history, and the like. The tracking program monitors which of the various links are selected and provides this information to the server. As discussed above, other available client information may also be sent to the server. This information is sorted and stored in the server database and may be analyzed manually or automatically.

The tracked information may be used to assemble resources geared toward the user's interests. Based upon the historic user profiles created in the server database, downloading of information to the same client on a subsequent visit to the same or different Web page may be done on a more intelligent basis. For example, users who have previously expressed an interest in sports-related trivia (as indicated by their previously tracked behavior) may be served with information targeted to audiences interested in sports. Similarly, users who have expressed greater interest in technology may be served with technology-related information that would be of much less interest to other users. The assembly of a resource such as a Web page may be easily accomplished. For example, the HTML document of the Web page may include a plurality of embedded resources. Previous choices made by a user on a particular client computer and stored in a user profile database may be used to determine which of the resources is to be downloaded to that client using simple logical processing instructions. For instance, a user profile which indicates that a user has a greater interest in sports-related information than in historical information may be used to download sports-related resources, such as GIF-type images and advertisements. Since the user has previously expressed a greater interest in sports, sports-related advertisements may therefore be targeted to that user.

A particular implementation of this mechanism is illustrated in FIG. 6. A Web page is requested by the client from Server A (**S601**). This Web page contains text, as well as embedded images which must be fetched from Server A (**S602**) and rendered (**S605**). In addition, the Web page contains embedded URLs that point to two resources on Server B. The first resource is a first CGI script 1, which is embedded inside the Web page using the standard HTML<IMG> tag (**S603**). In attempting to render the Web page, the client will automatically fetch the resource associated with the <IMG> tag on Server B, which will result in execution of the CGI script 1. This CGI script 1 can capture client information such as Network ID or Client ID (**S603A**). The CGI script also returns a transparent image (**S603B**).

The other resource on Server B is a Java applet, which is a combination ad banner and tracking program. This may be stored on any server. In attempting to render the Web page, the client will automatically fetch the Java code (**S604**), download, initialize, and start operation of the applet (**S607**, **S608**). After the applet is initialized, it contacts Server B to obtain other resources it needs in order to display images, play sounds, or control its overall look and behavior. In fact, the applet may obtain these resources by executing one or more CGI scripts or other processes that reside on Server B or elsewhere (**S607**). Based on information provided to these scripts through standard HTTP methods, including client information (**S607A**), such as network and client IDs, any other information such as the URL of the Web page, as well as information captured by the CGI script 1, and the previously constructed historical database profile (**S607B**), different information (images, sounds, text, etc.) may be returned to the applet. Such information can therefore be selected by the scripts based on Network and/or Client ID, the URL of the Web page, and the previously constructed client profile. This may be accomplished in the manner described above.

The STOP method of the applet which is executed, for example, when the user leaves the Web page (**S609**), will compute the difference between the current time and the time noted during execution of the START method. This difference, which is the time between execution of the STOP and execution of the START methods, is sent to the Server B for storage and analysis (**S610**). The information can be sent using standard JAVA network methods, such as opening a URL connection to a second CGI script on Server B designed to capture the tracked information (**S610A**, **S610B**). In step **S610A**, the second CGI script may obtain any information acquired by the tracking program (i.e., the JAVA applet), as well as client identifying indicia transmitted by the client, such as in the HTTP request header. This information can be stored in a database on Server B. If necessary, the information stored by both scripts may be combined into one more completed databases.

In this embodiment of the present invention, two distinct databases may be created. The first database is indexable by resource identity (such as URL), and includes information such as URL of the Web document, number of times accessed, identity of clients that accessed the Web document, amount of time displayed, amount of data displayed, average time displayed, number of times accessed, and the like. In the case of an ad banner or other embedded resource which may be accessed by a link made by a user while browsing another resource, the database may include additional information such as "click-through rate" (the number of times the ad banner was clicked on to go to the Web site of the advertiser), and the like.

A second database that may be created is indexable by individual client, and includes information concerning individual client's interests and preferences. These separate databases may be combined in a single database indexable by client or resource identity.

In another embodiment, illustrated in FIG. 7, the tracking program is used to create a database of information about a Web site (or, if desired, across multiple Web sites on multiple servers). In this case, the same tracking program is embedded in multiple Web pages served up by the same Server A. The tracking program in general originates from a Server B (but may also originate from Server A). The tracking program will monitor the time the Web page was displayed, and may capture any other information available to it. For example, the tracking program can determine the URL of the Web page it is embedded in and may determine the amount of information downloaded by the client.

In particular, a Web page is requested by the client from Server A (**S701**). This Web page contains text, as well as embedded images which must be fetched from Server A (**S702**) and rendered (**S705**). In addition, the Web page contains embedded URLs that point to two resources on Server B. The first resource is a CGI script, which is embedded inside the Web page using the standard HTML<IMG> tag (**S703**). In attempting to render the Web page, the client will automatically fetch the resource on

Server B, which will result in execution of a CGI script 1. This CGI script 1 can capture client information such as Network ID or Client ID (S703A) and returns a transparent image (S703B). The other resource on Server B is a Java applet. This may be stored on any server. In attempting to render the Web page, the client will automatically fetch the JAVA code, store it in RAM, initialize, and start operation of the applet (S707). The START method of the applet is executed and the applet takes note of the current time (S708). Thereafter, the applet contacts the Server A and, if security restrictions allow it, the applet queries the Server A for the page it is embedded in, determines its size, as well as the URLs of other embedded resources (such as images or video), and requests header information about these resources in order to determine their size (S709). In this case, the tracking program may determine the size of the fully rendered Web page, (i.e., the number of bits that must be downloaded in order to fully-render the Web page). If the tracking program is part of a larger embedded application that displays information downloaded from a server (such as a live news feed applet), the tracking program can also monitor the amount of information downloaded and displayed by the applet. Before or as the user leaves the Web page (S710), the tracking program can transmit this information to Server B for storage and analysis (S711, S711A, S711B). In this manner, it is possible to build a database of accurate information concerning how often different pages of a Web site are requested, how long they are displayed, and how much information was downloaded. This information would be of use to Web site administrators in order to judge the popularity of different Web pages, as well as for example to set advertising rates for any embedded advertisements.

In another embodiment, illustrated in FIG. **8**, a client **815** requests an HTML document **820** from Server A1 **800**. The HTML document **820** has an embedded reference to a resource. The resource comprises two components, each of which are stored on Server A2 **805**. Once the client has requested the HTML document **820** from Server A1 **800**, the client downloads an executable program **825** from Server B **810**. While this embodiment illustrates the case where the executable program is stored on Server B **810**, it will be understood by those skilled in the art that the executable program may also reside on Server A1 **800**, Server A2 **805**, or any other server coupled to the client **815**. Alternatively, the executable program may reside on the client **815** and thus need not be downloaded before being run on the client **815**. In a preferred embodiment, the executable program is an applet.

Once the executable program has been downloaded **825**, the client program retrieves the client ID **830**. Next, the client retrieves the resource ID **835**. The client then requests the first component of the resource **840**, which is stored on Server A2 **805**. The client **815** then renders and/or executes the first component **850** of the resource, allowing for user interaction with the first component. The executable program tracks the user's interaction with the first component **845**. The client **815** requests the second component of the resource **855** and the executable program tracks the user's interaction with the second component of the resource **860**.

Once the user interaction is complete **865**, the executable program packages the client ID, the resource ID, and the resource use data **870**. This packaged data is then sent **880** to Server B **810** for storage and analysis.

While this embodiment illustrates an example wherein both the first and second components of the resource are located on the same server, A2 **805**, those skilled in the art will recognize that the individual components of a resource need not be stored on the same server.

In another embodiment, illustrated in FIG. **9**, a client **915** requests an HTML document **920** from a Server A1 **900**. The HTML document has an embedded reference to a resource comprising three components: a component for displaying images, a component for rendering animations, and a component for providing form fill-in functionality.

After the client **915** has requested from Server A1 **900** the HTML document with an embedded reference to a resource, the client **915** downloads from Server B **910** an executable program **925**. The client **915** then executes a script **930** embedded in the HTML document requested from Server A1 **900**. Meanwhile, the executable program retrieves the client ID **935** for the client **915** upon which the executable program is running.

In the example illustrated in FIG. **9**, the script associated with the downloaded HTML document indicates that the first component of the referenced resource is one which displays images on the client **940**. Since this component is stored on Server A2 **905**, the client **915** downloads the functionality and data required to execute the first component of the resource **940**.

For purposes of this example, it is assumed that the first component of the embedded resource allows for user interaction. As the user interacts with the first component, the component keeps track of the user's interaction. At some point, the user interaction may determine that it is appropriate to start the second component of the embedded resource. Since the second component is stored on Server A2 **905**, the client **915** next downloads the second component of the resource **945**. In this example, the second component of the embedded resource provides animation functionality to the client. The second component, thus downloaded and functioning **950** on the client **915**, tracks user interaction with the second component.

When the user interaction with the second component so indicates, via the logic of the script associated with the HTML document **930**, the client will download the next component of the resource; in this example, the third component of the resource is the functionality to fill in forms. Thus, the client **915** downloads from Server A2 **905** the resource's component which provides form fill-in functionality **955**. Once this third component is downloaded, the client **915** renders the form **960** and allows user interaction for filling in the form **965**. The form fill-in component tracks this user interaction with the component.

As each component collects user tracking information, the information is retrieved by the executable program **975**.

At some point, the user's interaction will indicate that the user has left the HTML document which contained the resource for which the three components were required **970**. The executable program, having recognized that the user has left the HTML document embedded with the resource, tags all of the retrieved tracking information from the components with client ID and resource ID information **980**. The appropriately tagged tracking information is then sent by the client program **980** to Server B **910**. On Server B **910**, there is running a script **985** which captures the executable program data sent from the client **915** and stores it in database **990** for storage and possible later analysis.

While the example illustrated in FIG. **9** has been described as one in which the individual components of the resource do the tracking of the client interaction such that the client interaction tracking data generated by each of the components is in the proper format, except for being tagged with appropriate ID tags, for storage in the database **990** stored on Sever B **910**, those skilled in the art will recognize

that the components may be designed such that some or all of them do not generate data about each user interaction which is usable in the database **990**, but rather pass the interaction data to the executable program **975**. The executable program then must generate data pertaining to the user interactions in a format suitable for storage in the database **990**.

The example illustrated in FIG. **10** is closely associated with the one illustrated in FIG. **9**; the point of difference between the two is that in the example illustrated in FIG. **10**, the executable program may decide at a given time, by way of example: based on time, user interaction, request of Server B **910**, or instructions contained in the HTML document, to collect tracking information from each of the loaded components **1075**. When the executable program decides to collect the interaction information, it will then tag all of the collected tracking information with ID data **1080** and send the tagged information to Server B **910** for storage in the database **990**. That is, in the example illustrated in FIG. **10**, the executable program does not necessarily wait for the user to leave the HTML document with the embedded resource before collecting the interaction information, tagging it with ID information, and sending it to Server B **910**; the executable program may collect the interaction information **1075**, tag it with ID information **1080**, and send it to Server B **990** based on an event other than the user leaving the HTML document with the embedded resource.

As with the example illustrated in FIG. **9**, the script associated with the downloaded HTML document **930** indicates that the first component of the referenced resource is one which displays images on the client **940**. Since this component is stored on Server A2 **905**, the client **915** downloads the functionality and data required to execute the first component of the resource **940**.

As with the example illustrated in FIG. **9**, it is assumed that the first component of the embedded resource allows for user interaction. As the user interacts with the first component, the component keeps track of the user's interaction. At some point, the user interaction may determine that it is appropriate to start the second component of the embedded resource. Since the second component is stored on Server A2 **905**, the client **915** next downloads the second component of the resource **945**. As in the example illustrated in FIG. **9**, the second component of the embedded resource provides animation functionality to the client. The second component, thus downloaded and functioning **950** on the client **915**, tracks user interaction with the second component.

When the user interaction with the second component so indicates, via the logic of the script associated with the HTML document **930**, the client will download the next component of the resource; in this example the third component of the resource is the functionality to fill in forms. Thus, the client **915** downloads from Server A2 **905** the resource's component which provides form fill-in functionality **955**. Once this third component is downloaded, the client **915** renders the form **960** and allows user interaction for filling in the form. The form fill-in component tracks user interaction with the component.

At some point, based on, for example and not by way of limitation, the time, a request from Server B **910**, a particular user interaction, or instructions contained in the HTML document in which the resource is embedded, the executable program collects tracking information from each of the loaded components **1075**. The executable program then tags the collected tracking information with client ID and resource ID information and sends the appropriately tagged tracking information **1080** to Server B **910**. On Server B **910**, there is running a script which captures the executable program data sent from the client **915** and stores it in database **990** for storage and possible later analysis.

In another embodiment, illustrated in FIG. **11**, a client **1115** requests an HTML document **1120** from a Server A1 **1100**. The HTML document has an embedded reference to a resource comprising three components: a component for displaying images, a component for rendering animations, and a component for providing form fill-in functionality. In this example, we are interested in collecting interaction data about only two of the three components which comprise the resource: the component providing animation functionality and the component providing form fill-in functionality.

After the client **1115** has requested from Server A1 **1100** the HTML document with an embedded reference to a resource, the client **1115** downloads from Server B **1110** an executable program **1125**. The client **1115** then executes a script **1130** embedded in the HTML document requested from Server A1 **1100**. Meanwhile, the executable program retrieves the client ID **1135** for the client **1115** upon which the executable program is running.

In the example illustrated in FIG. **11**, the script associated with the downloaded HTML document indicates that the first component of the referenced resource is one which displays images on the client **1140**. Since this component is stored on Server A2 **1105**, the client **1115** downloads the functionality and data required to execute the first component of the resource **1140**.

For purposes of this example, it is assumed that the first component of the embedded resource allows for user interaction; however, since in this embodiment we are not interested in tracking user interaction with the first component, the first component does not track user interactions. At some point, the user interaction may determine that it is appropriate to start the second component of the embedded resource. Since the second component is stored on Server A2 **1105**, the client **1115** next downloads the second component of the resource **1145**. In this example, the second component of the embedded resource provides animation functionality to the client. The second component, thus downloaded and functioning **1150** on the client **1115**, tracks user interaction with the second component. As the second component tracks user interactions, it may decide to send user interaction information to Server B **1110**. If so, the second component will retrieve ID-tagging data from the executable program and appropriately tag the user interaction information with the ID-tagging data **1172**. The second component will then send the tagged user interaction information to Server B **1110** for storage in database **1190** and possible later analysis.

When the user interaction with the second component so indicates, via the logic of the script associated with the HTML document, the client will download the next component of the resource; in this example the third component of the resource is the functionality to fill in forms. Thus, the client **1115** downloads from Server A2 **1105** the resource's component which provides form fill-in functionality **1155**. Once this third component is downloaded, the client **1115** renders the form **1160** and allows user interaction for filling in the form. The form fill-in component tracks this user interaction with the component.

As with the second component, the third component tracks user interactions and may decide to send user interaction information to Server B **1110**. If so, the third com-

ponent will retrieve ID-tagging data from the executable program and appropriately tag the user interaction information with the ID-tagging data **1170**. The third component will then send the tagged user interaction information to Server B **1110** for storage in database **1190** and possible later analysis.

In yet another embodiment, the tracking program is used to assemble a bill for the user's access to information. For example, users who have access to a live news or entertainment feed may be charged according to the amount of information displayed, either according to bit size or time, or both. Imagine that the tracking program is attached to a live feed applet. The tracking program monitors the time the information is displayed and the amount of bits downloaded and automatically transmits this information back to a server when the user leaves. Together with the user's ID (client and network), and billing information that the user was previously requested to enter, it is possible to determine the correct charge for the user. Similarly, a user could be charged and billed for time spent on a Web page, as well as amount of information downloaded by him or her.

The methods embodied in the invention may be used to create web resources with so-called "persistent" state. That is, the tracking program, in addition to the client profile database, may also be used to create a Web resource that appears to automatically "remember" the user's previous interactions on the Web resource. This may be implemented as in FIG. **6**. For example, consider a Web page with an embedded Crossword program which also incorporates tracking mechanisms. When the page is rendered and the Crossword program commences, a user is able to use the keyboard and mouse to fill in letters on the Web page based on clues that are displayed. At the same time, these choices are tracked, along with any other information including but not limited to time. Before or at the time the user leaves the Web page, the tracked information is sent to a server for storage (**S610**). When the user later returns to that page, the network or client ID is used to automatically fill in the letters in the crossword that were previously selected (As in **S607**–**607**C).

Although the invention has been described in terms of preferred embodiments, those skilled in the art will recognize that various modifications of the invention can be practiced within the spirit and scope of the appended claims. Thus, for example, the scripts used to transfer data need not be CGI scripts but could be a dedicated server or a direct connection to the database, such as using JDBC (Java Database Connectivity) to place data into the database.

In addition, while the preferred embodiments have been described in connection with JAVA applets that are executable on a client, the tracking of user interaction may be accomplished by a client executable program written in a language other than JAVA. For example, the teachings of the present invention may be accomplished using Active-X components in conjunction with the Internet Explorer Web browser. In addition, the tracking program need not be a program that executes on the client computer. For example, the tracking program may comprise a CGI script located on a server. Upon execution of the CGI script, the time at which a Web page is downloaded may be determined. By modifying Web browser software using appropriate instructions, the browser can be used to send a signal to the server that downloaded the Web page upon the occurrence of a predetermined user operation (such as exiting the Web page or clicking on a link to another Web page or resource). In this manner, a program running on the server can be used to determine the total time period the user has interacted with and displayed the Web page.

It should also be appreciated that while the preferred embodiments of the tracking program use a single database to store the information, multiple databases could be used to store and process the information.

In addition, while in the preferred embodiments of the tracking program the server that originated the tracking program and the database reside on the same machine, this is not a requirement of the present invention. The database may instead reside on a separate machine from that which serves the tracking program. Similarly, while in the preferred embodiments the server that originates the network resource, or Web page (Server A), and the server that originates the tracking program (Server B) are different servers, this is not a requirement of the present invention. The network resource (Web page) and the tracking program may be served out by the same server.

It should also be appreciated that while in the preferred embodiments the tracking program uses the HTTP and TCP/IP protocols, other network data transmission protocols could be used that implement the same functionality. Moreover, use of an HTML formatted Web page is not necessary. The information supplied to the user may not be in the form of an HTML or Web document such as a Web page, but can be some other form of information. In addition, the tracking program need not be downloaded to the client from the server, but can be an added module to the client application or Web browser running on the client, or may be stored elsewhere on the client machine. For example, in the former case, added modules could be plug-ins and in the latter case could be referred to as cached resources. In such cases, the client application or Web browser would include appropriate means to enable activation of the tracking program and the uploading of a client profile based upon the user's interaction with a Web page or network resource.

Moreover, although in the preferred embodiments it is envisioned that the network resource or Web page is downloaded from a remote server, this is not a limitation of the invention. The precise location of the target document or server is not important. For example, the target document may even be located on the hard drive of the client machine.

Also, while in the above-described embodiments, the client profile is created automatically using information acquired by the tracking program and one or more CGI scripts and is stored in the server database, the client profile can be created in a different manner and/or supplemented by additional information. For example, one such technique for creating a client profile is through the use of HTML "fill-in" form tags. In such cases, the client profile is created not by the tracking program, but instead by the client. Based on the client profile, the server can serve out information targeted to the client's interest, as revealed by the fill-in form.

Also, while the preferred embodiments have been described in the context of Web browser software, the techniques of the invention apply equally whether the user accesses a local area network, a wide area network, a public network, a private network, the Internet, the World Wide Web, or the like, and whether access to the network is achieved using a direct connection or an indirect connection. For example, in connection with the World Wide Web, the teachings of the present invention apply whether a network connection is obtained via a direct Internet connection or indirectly through some on-line service provider. Thus, the "computer network" in which the invention is implemented should be broadly construed to include any computer network in which one or more clients is connectable to one or

more servers, including those networks based upon the client-server model in which a client can link to a "remote" document (even if that document is available on the same machine, system, or "Intranet").

It should also be appreciated that while in the preferred embodiments the tracking program is downloaded with the Web page from the server, this is not a limitation of the invention. The tracking program need not be embedded within an existing Web page, but rather may be embedded within a Web browser or supported elsewhere within the client itself. Thus, the tracking program may be initiated whenever a call to a Web page or network resource is made, such as when a search to a particular URL is initiated, or when a previously-stored URL is launched.

We claim:

1. In a network having one or more servers connectable to one or more clients, a method of monitoring interaction through a first client of the one or more clients with a first resource, the first resource comprising a plurality of components and having a resource identifier associated therewith, the method comprising:

downloading to the first client a first component of the first resource from a first server;

executing on the first client an executable program to download one or more second components of the first resource;

monitoring interaction through the first client with the first component of the first resource and with the one or more second components of the first resource;

generating resource use data; and

transmitting from the first client to a second server information comprising resource use data and the resource identifier.

2. The method of claim 1, wherein the monitoring is performed by one or more executable elements of the one or more second components.

3. The method of claim 2, wherein the one or more executable elements transfer resource use data to the executable program upon use of the one or more second components.

4. The method of claim 2, wherein the one or more executable elements store resource use data upon use of the one or more second components and transfer the stored resource use data to the executable program upon receipt of a request from the executable program.

5. The method of claim 2, comprising the executable program transferring the resource identifier to the one or more executable elements, and wherein the step of transmitting the resource use data and resource identifier is performed by the one or more executable elements.

6. The method of claim 1 wherein the resource use data comprises data generated in response to predetermined interactions with the first component of the first resource and predetermined interactions with the one or more second components of the first resource.

7. The method of claim 6 wherein the predetermined interactions with the first component comprise interactions with the first component other than interactions with the first component used by the first component or the one or more second components.

8. The method of claim 6 wherein the predetermined interactions with the one or more second components comprise interactions with the one or more second components other than interactions with the one or more second components used by the first component or by the one or more second components.

9. The method of claim 1, wherein the step of transmitting the resource use data and resource identifier is performed by the executable program.

10. The method of claim 9, wherein the monitoring is performed by one or more executable elements of the one or more second resources, and comprising the one or more executable elements transferring the resource use data to the executable program.

11. The method of claim 1 wherein the resource use data comprises data other than data used by the first component.

12. The method of claim 1 wherein the resource use data comprises data other than data used by the second or more components.

13. The method of claim 1 wherein the resource use data comprises data other then data used by the first resource.

14. The method of claim 1 wherein the first program is downloaded from the first server.

15. The method of claim 1 wherein the first program is downloaded to the first client as part of the first component of the first resource.

16. The method of claim 1 wherein the first component of the first resource comprises the first program.

17. The method of claim 1, comprising downloading the executable program from a third server.

18. The method of claim 1, comprising downloading the executable program from the second server.

19. The method of claim 1, wherein the executable program is a part of the first component of the first resource.

20. The method of claim 1, wherein the executable program is implemented in the Java programming language.

21. The method of claim 1, wherein the one or more second components are implemented in the Java programming language.

22. The method of claim 1, wherein the resource use data comprises data representative of a duration of time for which the first resource was used by the first client.

23. The method of claim 1, wherein the resource use data comprises data representative of a duration of time for which the first component, or a part or parts thereof, was used by the first client.

24. The method of claim 1, wherein the resource use data comprises data representative of a duration of time for which the one or more second components, or a part or parts thereof, were used by the first client.

25. The method of claim 1, wherein the resource use data comprises data indicative of whether the first user used the resource.

26. The method of claim 1 wherein the network comprises a computer network.

27. The method of claim 1 wherein the network comprises a communications network.

28. In a computer network comprising one or more servers and one or more clients, a method of monitoring interaction through a first client of the one or more clients with a first resource or at least one component thereof obtained by the first client from at least one first server of the one or more servers, the method comprising:

downloading the first resource in a series of one or more components from the at least one first server to the first client;

downloading a tracking program from a server of the one or more servers to the first client, wherein the downloading of the first resource or at least one of the components thereof causes the downloading of the tracking program;

executing the tracking program on the client computer to monitor interaction through the client computer with at

least one of the first resource, at least one of the components thereof, and one or more second resources, the one or more second resources having been obtained by the first client from at least one server of the one or more servers as a result of interaction through the first client with at least one of the first resource, at least one of the components thereof, and a second resource of the one or more second resources;

storing resource use data associated with the monitored interaction; and

communicating the resource use data to a server of the one or more servers.

29. The method of claim **28**, wherein downloading the first resource comprises downloading a Web page.

30. The method of claim **29**, wherein the tracking program is embedded in the Web page.

31. The method of claim **28**, comprising storing the resource use data in a server of the one or more servers.

32. The method of claim **31**, comprising storing the resource use data in a database.

33. The method of claim **28**, wherein monitoring user interaction comprises monitoring input device events.

34. The method of claim **33**, wherein the input device events comprise at least one of keyboard events and mouse events.

35. The method of claim **28**, comprising downloading the first resource as a result of a request initiated by input from a user of the first client.

36. The method of claim **28**, comprising downloading the tracking program from a second server of the one or more servers.

37. The method of claim **28**, comprising monitoring details of choices made by a user of the first client using an input device of the first client, the choices being associated with at least one of the first resource, at least one of the one or more components thereof, and the one or more second resources.

38. The method of claim **28**, comprising storing the resource us data in the client computer.

39. The method of claim **28**, comprising communicating the resource use data from the first client to a server of the one or more servers.

40. In a network having one or more servers connectable to one or more clients, a method of monitoring interaction through a client of the one or more clients with a first resource which is stored on a first server and which may be stored in a cache, the method comprising:

downloading from one or more source servers one or more files containing embedded references to the first resource and a second resource;

the client requesting or using the first resource one or more times from the first server or from the cache;

downloading the second resource from a second server to the client, the first server and second server comprising two servers and the second resource not being part of the first resource; and

using the second resource to monitor interaction through the client with the first resource.

41. The method of claim **40**, wherein the second resource is an applet.

42. The method of claim **41**, wherein the applet is implemented in the Java programming language.

43. The method of claim **40**, wherein the step of requesting or using one or more files comprises requesting from one or more source servers a plurality of files each containing embedded references to the first resource and a second resource.

44. The method of claim **43**, wherein the step of the client requesting or using the first resource comprises the client retrieving the first resource from cache for one or more files following a first file in the plurality of files.

45. The method of claim **43**, wherein the step of requesting a plurality of files comprises requesting the files from a plurality of source servers.

46. The method of claim **40**, wherein the step of requested the second resource comprises requesting the second resource each time the one or more files is requested.

47. The method of claim **46**, wherein the step of using the second resource to monitor the number of times the first resource is requested or used comprises monitoring at the second server the number of times the second resource is requested to thereby monitor the number of times a file containing an embedded reference to the first resource is requested or used.

48. The method of claim **40**, wherein the second resource is an executable program, comprising using the second resource to monitor a number of times the first resource is requested or used, wherein using the second resource comprises executing the second resource on the client to monitor each time the first resource is retrieved from the first server or from the cache.

49. The method of claim **40** wherein the network comprises a computer network.

50. The method of claim **40** wherein the network comprises a communications network.

51. In a network having one or more servers connectable to one or more clients, a method of monitoring interaction through a first client of the one or more clients with a first resource by a user, the first resource comprising a plurality of components and having a resource identifier associated therewith, the method comprising:

downloading to the first client a first component of the first resource from a first server;

executing on the first client an executable program to download one or more second components of the first resource;

monitoring interaction through the first client with the first component of the first resource and interaction through the first client with the one or more second components of the first resource;

generating resource use data; and

transmitting from the first client to a second server information comprising resource use data and the resource identifier.

52. The method of claim **51** wherein the resource use data comprises data representative of a plurality of preferences of the user.

53. The method of claim **51** wherein the resource use data comprises data representative of a plurality of interests of the user.

54. The method of claim **51** wherein the resource use data comprises data which does not affect the user's contemporaneous experience of the resource.

55. The method of claim **51** wherein the step of generating resource use data does not require the active participation of a user.

56. The method of claim **51** wherein the step of generating resource use data comprises the automatic determination of a user's preferences.

57. In a network having one or more servers connectable to one or more clients, a method of monitoring interaction through a first client of the one or more clients with a first resource comprising a plurality of components, the first

resource having a resource identifier associated therewith, the method comprising:

downloading to the first client a first component of the first resource from a first server;

monitoring interaction through the first client with the first component of the first resource to generate resource use data; and

transmitting from the first client to a second server the resource use data and the resource identifier.

58. The method of claim **57**, comprising downloading the executable program from a third server.

59. In a network having one or more servers connectable to one or more clients, a method of monitoring interaction through a first client of the one or more clients with a first resource comprising a plurality of components, the first resource having a resource identifier associated therewith, the method comprising:

downloading to the first client a first component of the first resource from a first server;

executing on the first client an executable program to download one or more second components of the first resource;

monitoring interaction through the first client with the first component of the first resource to generate resource use data;

monitoring interaction through the first client with the one or more second components to generate resource use data;

executing on the first client one or more of the second components to download one or more third components of a second resource;

monitoring interaction through the first client with the one or more third components to generate resource use data; and

transmitting from the first client to a second server the resource use data and the resource identifier.

* * * * *

US006516336B1

(12) **United States Patent**
Davis et al.

(10) **Patent No.: US 6,516,336 B1**
(45) **Date of Patent: Feb. 4, 2003**

(54) **METHOD AND SYSTEM FOR USING A TWO-TIERED CACHE**

(75) Inventors: **Charlotte Elizabeth Davis**, Chapel Hill, NC (US); **Bradford Austin Fisher**, Carrboro, NC (US); **Jonathan Scott Greenfield**, Holly Springs, NC (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/392,035**

(22) Filed: **Sep. 8, 1999**

(51) **Int. Cl.**[7] **G06F 15/14**
(52) **U.S. Cl.** **709/201**; 709/203; 709/220; 709/223; 707/8
(58) **Field of Search** 709/201, 203, 709/220, 223; 707/8

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,295,243	A *	3/1994	Robertson et al.	345/649
5,414,812	A *	5/1995	Filip et al.	707/103 R
5,630,125	A *	5/1997	Zellweger	707/102
6,052,720	A *	4/2000	Traversat et al.	709/220
6,105,063	A *	8/2000	Hayes, Jr.	709/222
6,259,451	B1 *	7/2001	Tesler	345/419

* cited by examiner

Primary Examiner—David Wiley
Assistant Examiner—Frantz B. Jean
(74) *Attorney, Agent, or Firm*—Jeanine S. Ray-Yarletts; Marcia L. Doubet

(57) **ABSTRACT**

A method, system, and computer program product code using a two-tiered cache for hierarchically structured data. The present invention significantly reduces the frequency of computationally intense processing used to retrieve hierarchically structured data and reduces the system cache storage requirements for maintaining coalesced images.

21 Claims, 7 Drawing Sheets

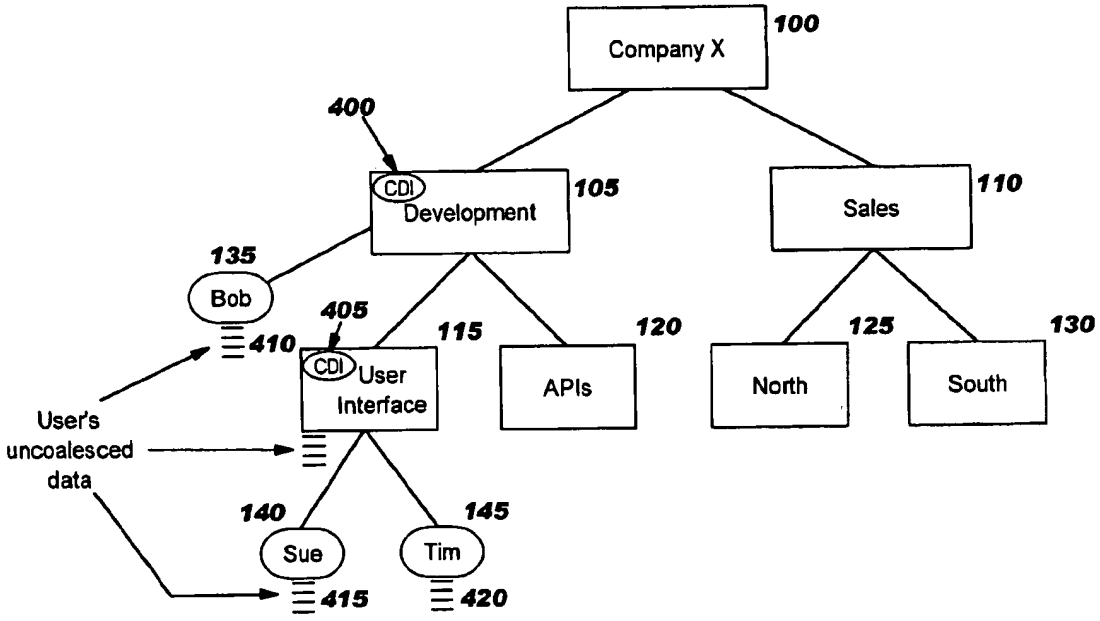
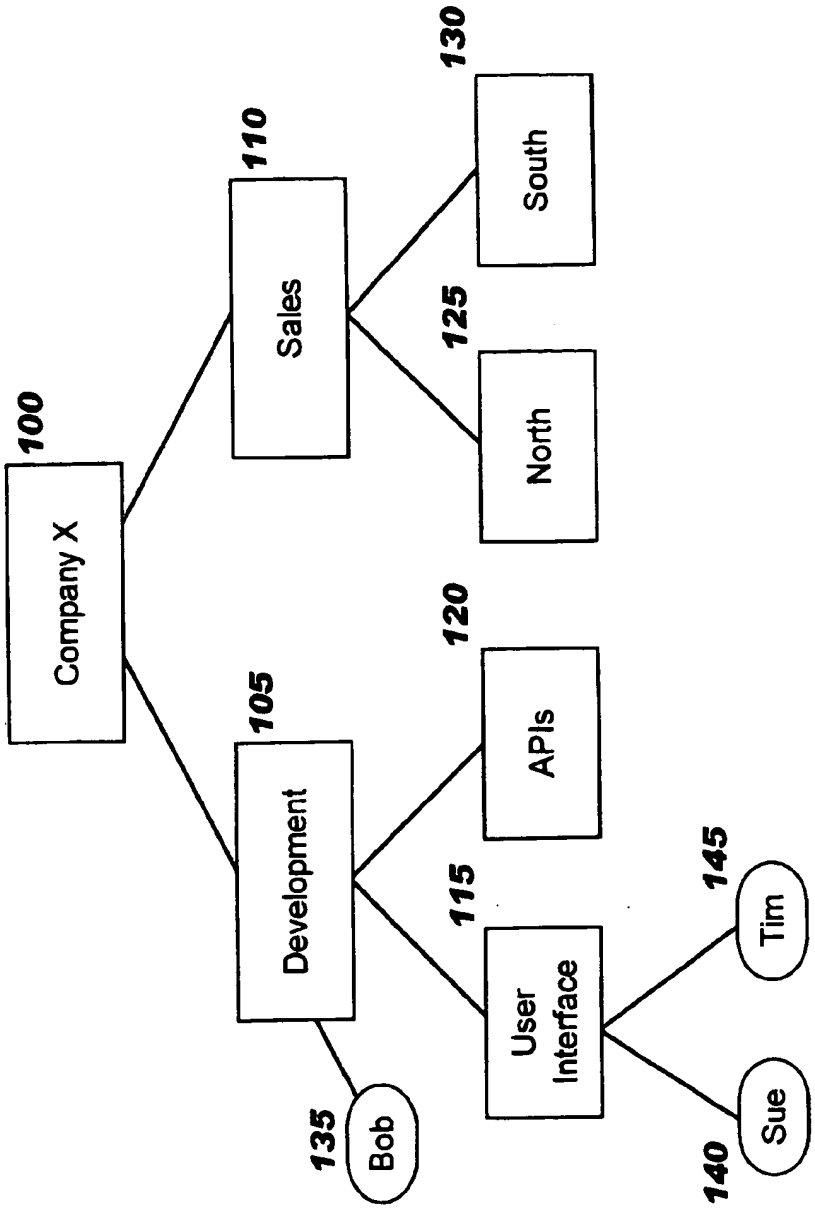


FIG. 1
Prior Art



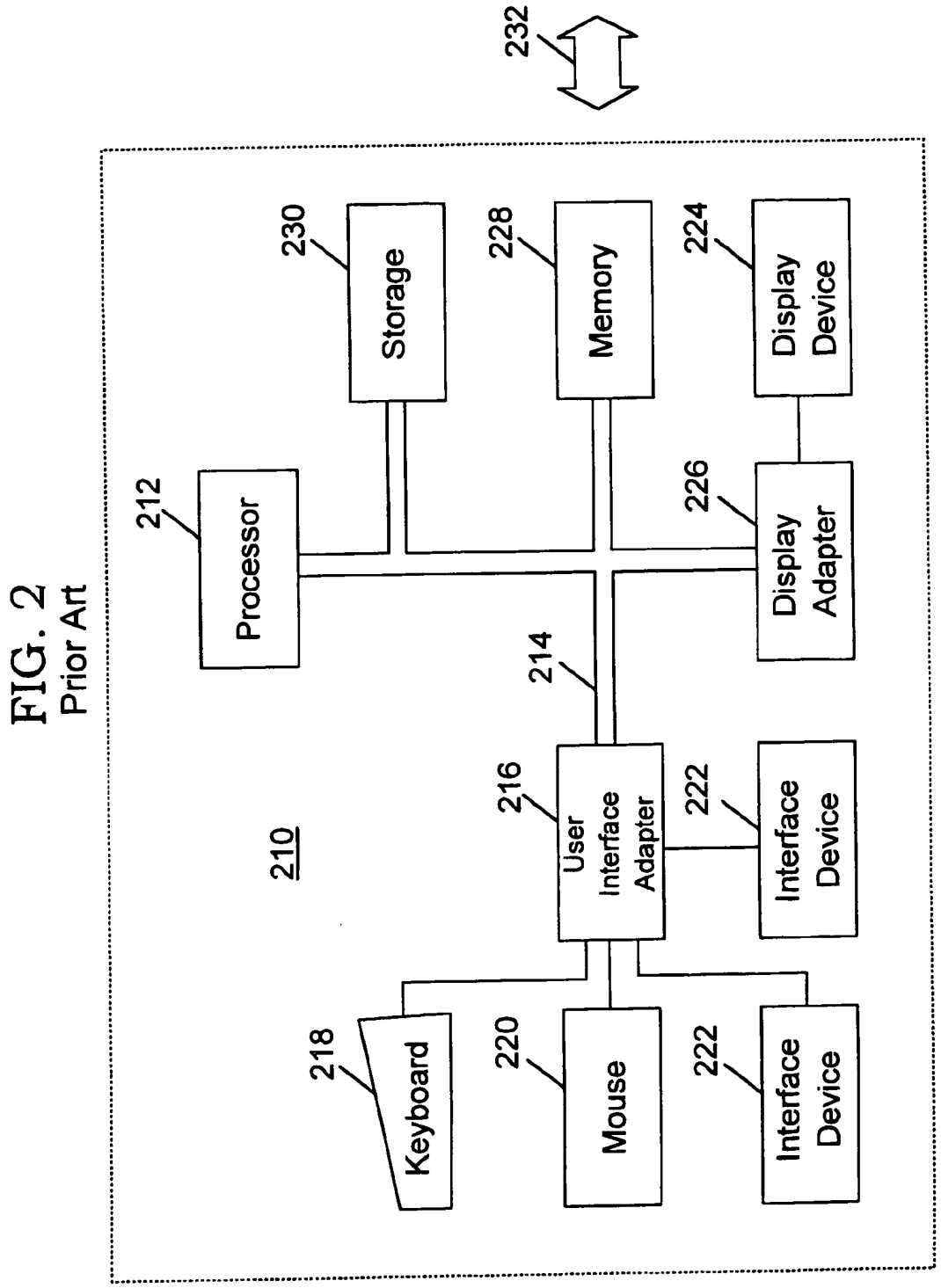


FIG. 2
Prior Art

FIG. 3
Prior Art

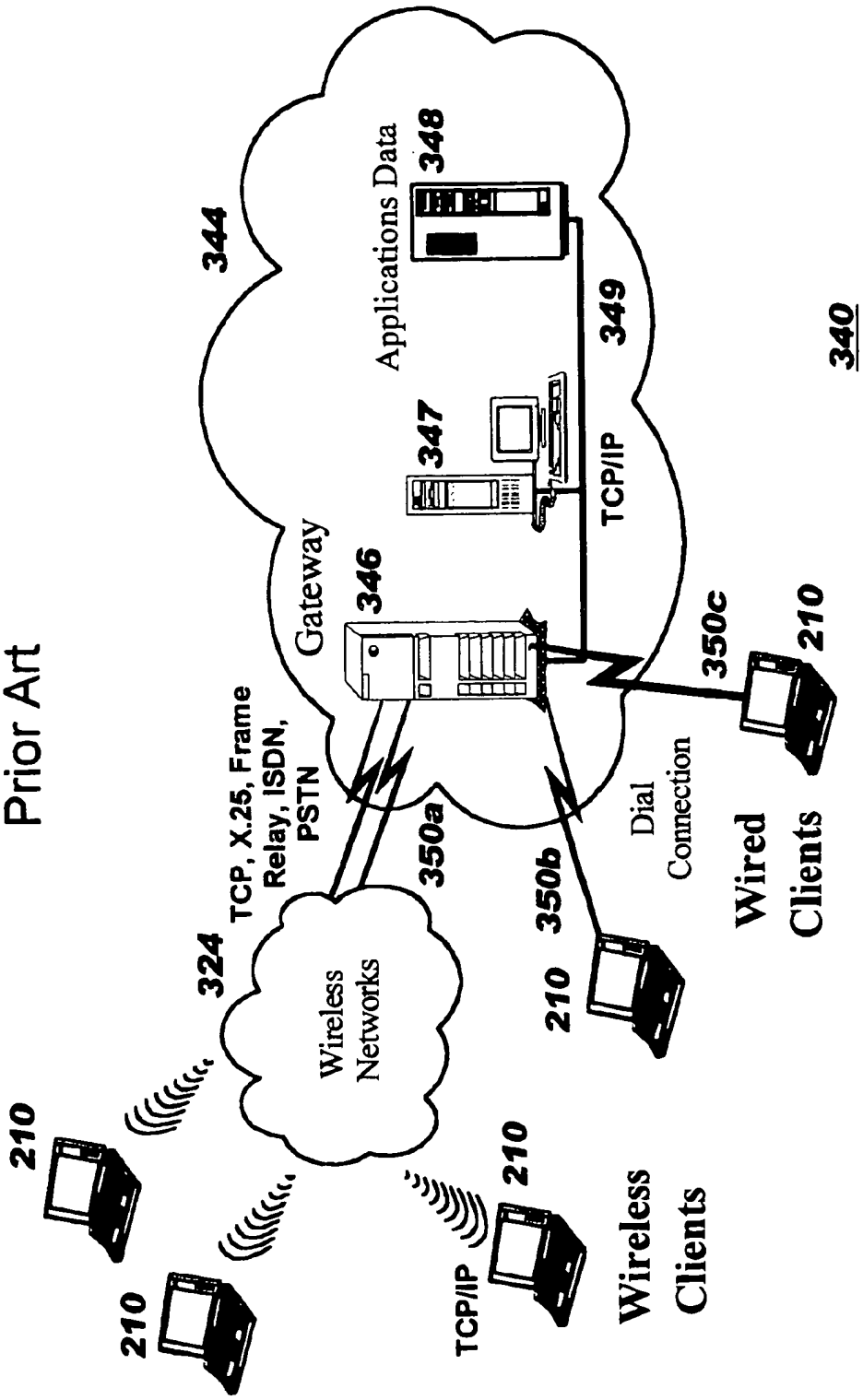


FIG. 4

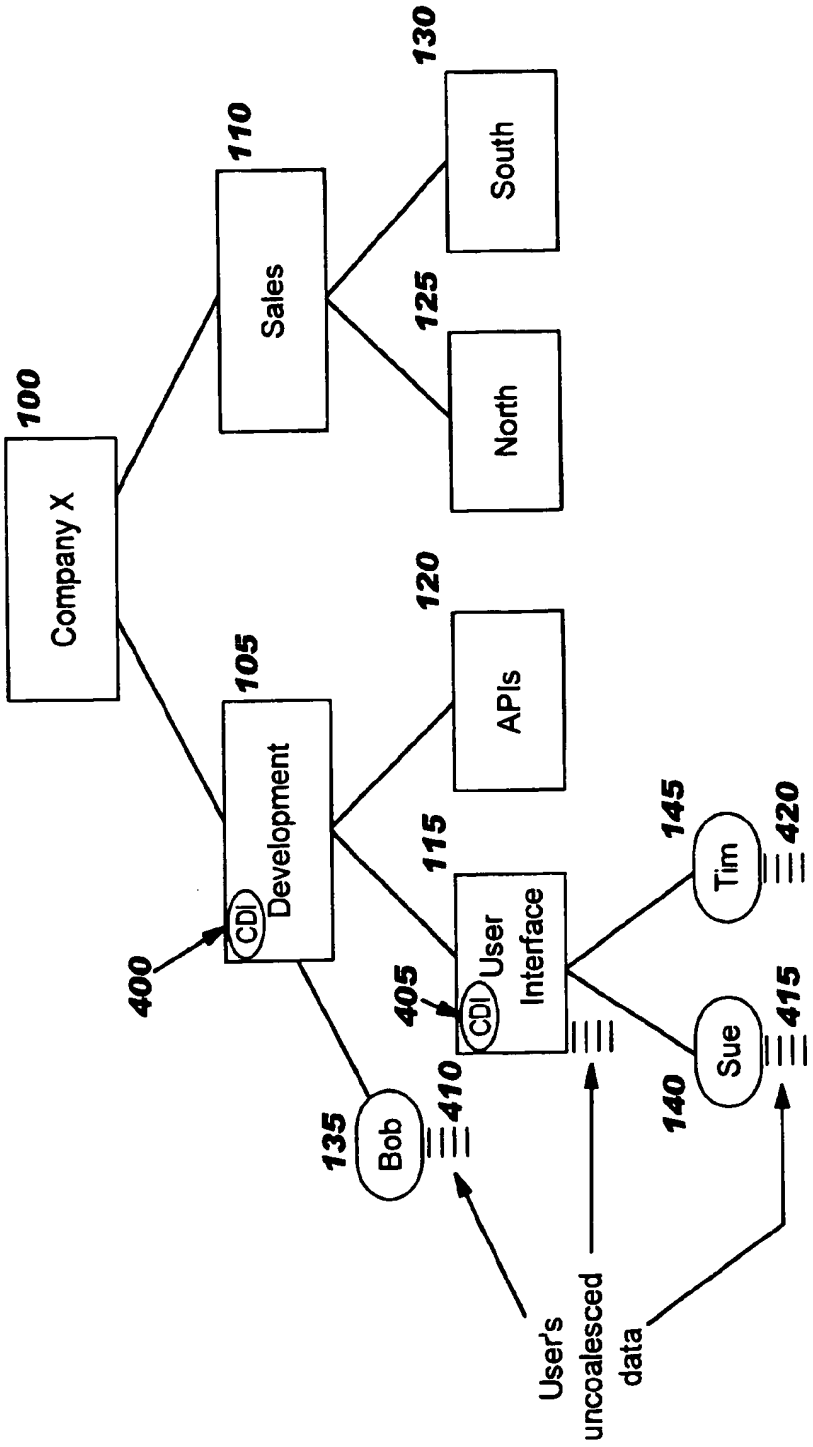

Company X
100
Sales
110
South
130
North
125
Development
105
APIs
120
User Interface
115
CDI
400
CDI
405
Bob
135
410
Sue
140
415
Tim
145
420
User's uncoalesced data

FIG. 5

User Requests to View Preferences Without Refresh

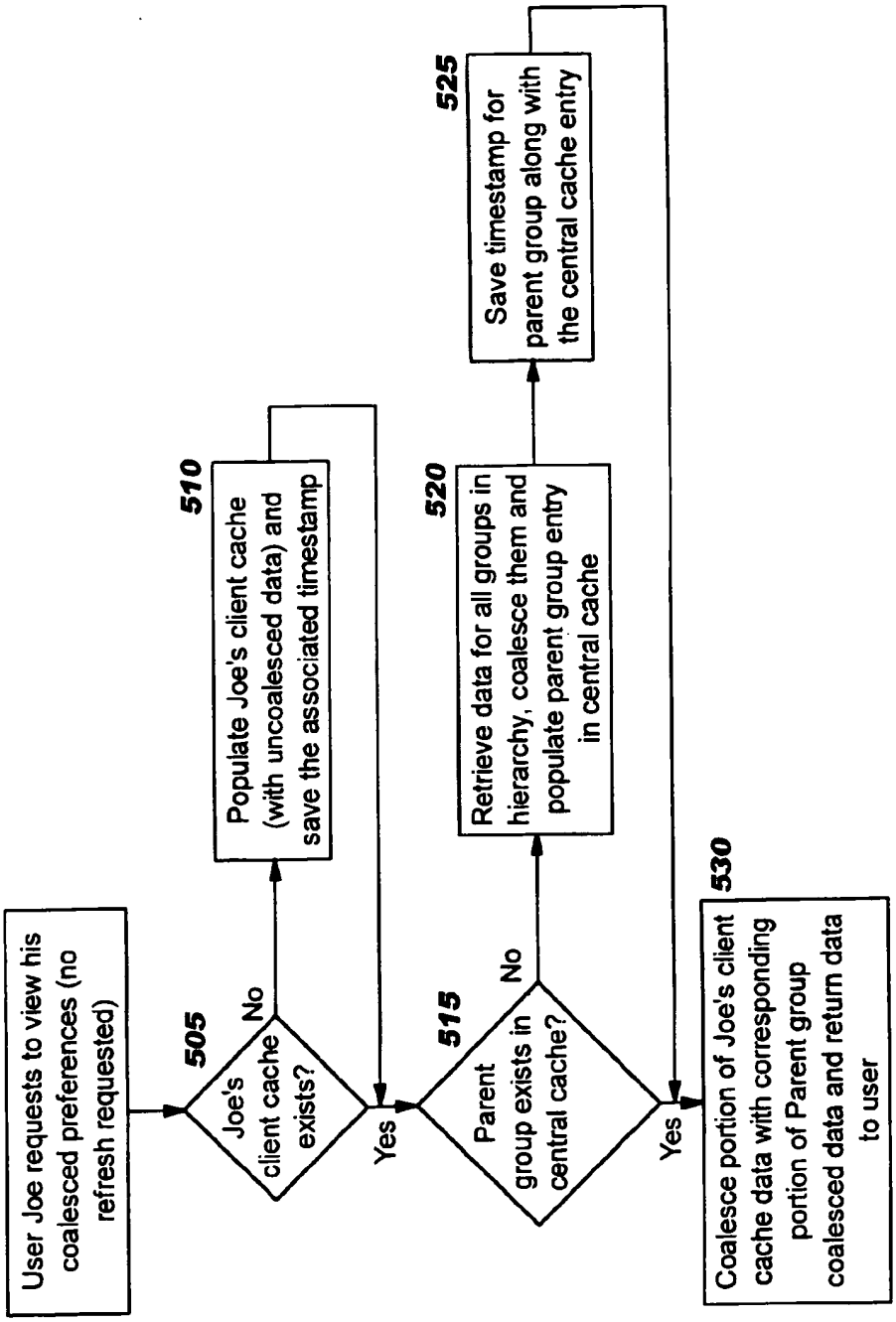


FIG. 6

User Requests to View Preferences With Refresh

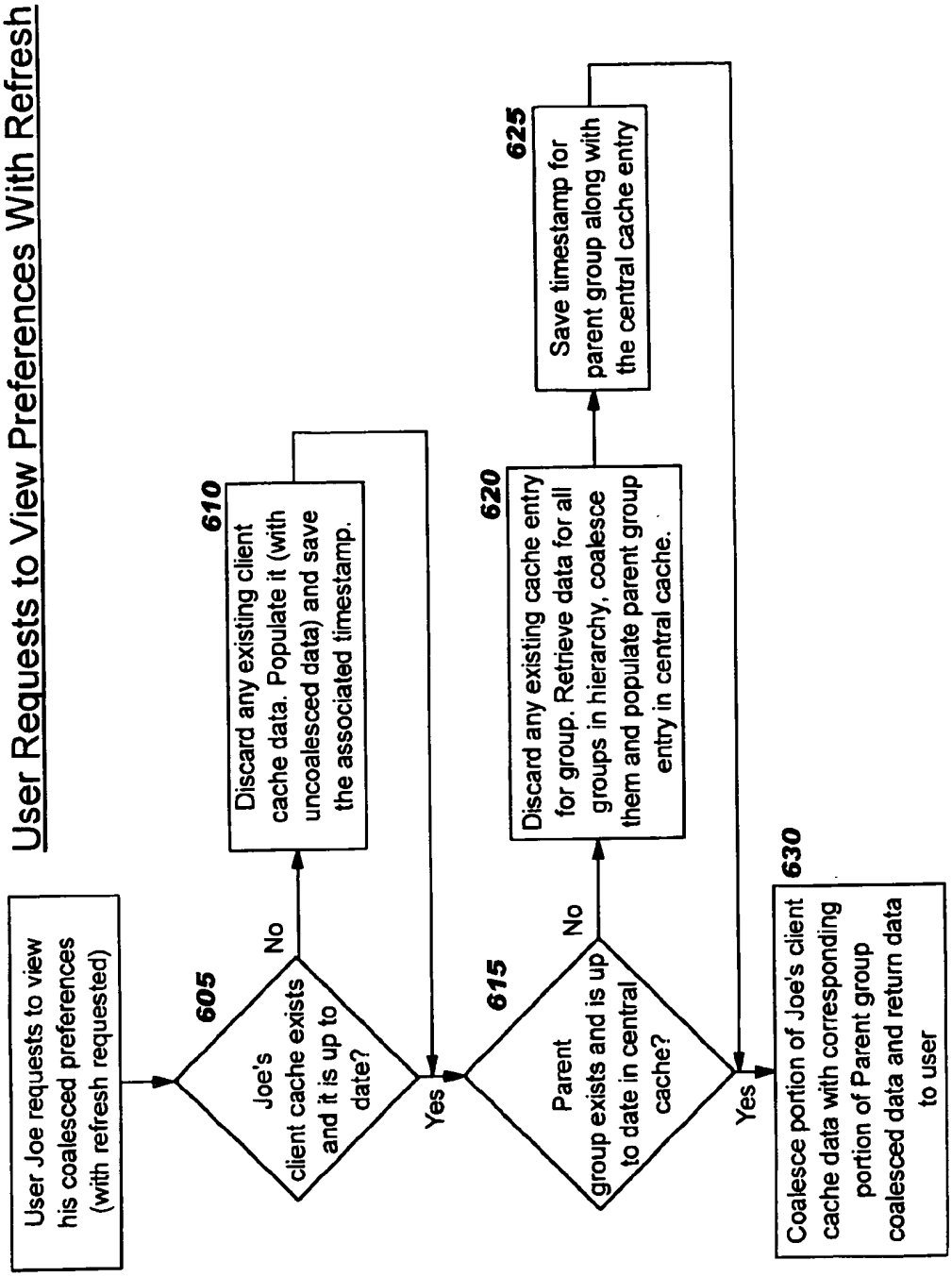
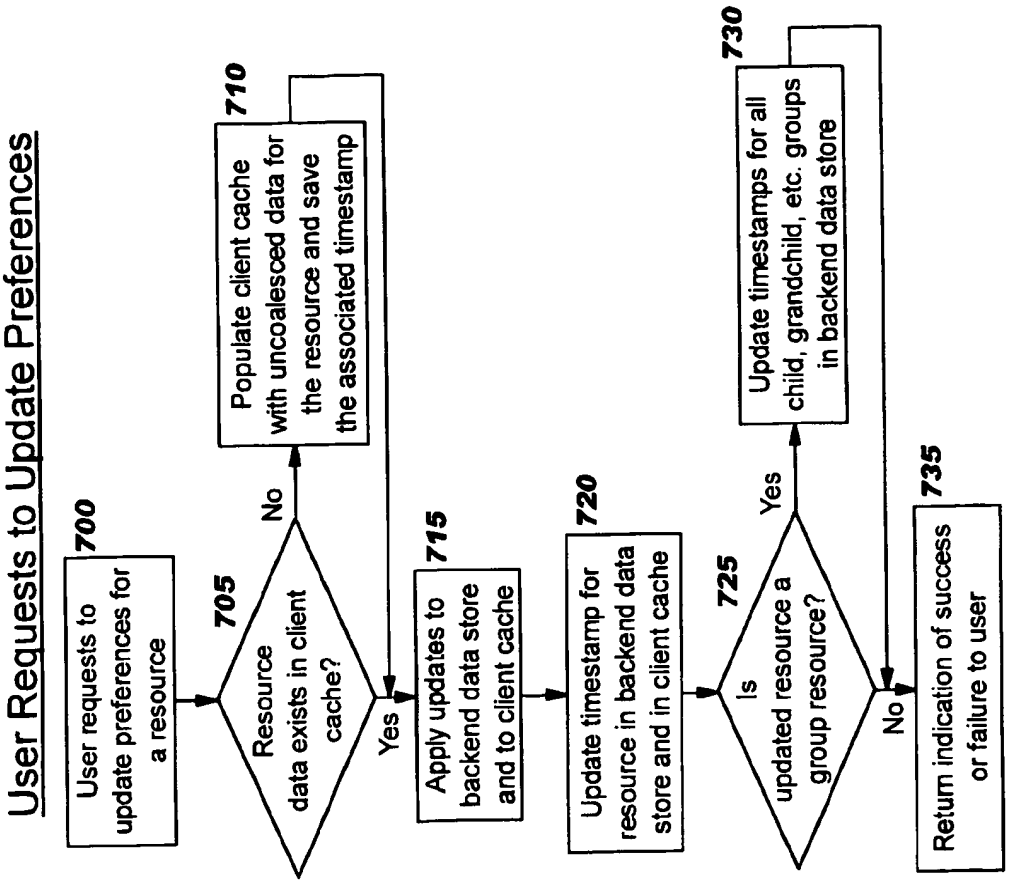


FIG. 7


User Requests to Update Preferences
700
User requests to update preferences for a resource
705
Resource data exists in client cache?
No
710
Populate client cache with uncoalesced data for the resource and save the associated timestamp
Yes
715
Apply updates to backend data store and to client cache
720
Update timestamp for resource in backend data store and in client cache
725
Is updated resource a group resource?
Yes
730
Update timestamps for all child, grandchild, etc. groups in backend data store
No
735
Return indication of success or failure to user

METHOD AND SYSTEM FOR USING A TWO-TIERED CACHE

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a computer system, and deals more particularly with a method, system, and computer program product for using a two-tiered cache for storing and accessing hierarchical data, in order to minimize memory usage and improve performance.

2. Description of the Related Art

In the distributed client/server environment, there is a need to disseminate information within a company or corporation that is required by users for everyday operation. This information typically includes such things as accounting information, computer configuration information, etc., and is typically available on one or more servers which are accessed by the users. Since companies, particularly large corporations, are often organized hierarchically (such as divisions, departments, etc.), each of these hierarchical levels may specify information pertinent to that level, where each lower level of hierarchical information augments or overrides the level(s) above it. In particular, this hierarchical approach to specifying information is often used for computer configuration information. In this scenario, each user of the company systems accesses the information provided by the higher levels of the hierarchy, for example to use when connecting to the company network or when installing software on his computer. The user at this point occupies the lowest level of the hierarchy, and may customize the information obtained from the hierarchy above him. These customized values are often referred to as "user preferences" and may be unique to any user within the hierarchy.

FIG. 1 is an example of this type of hierarchical structure. This structure represents hierarchically stored information, which typically mirrors the hierarchy of the organization. Company X **100** represents information stored for the corporate or highest level of the company. Any information defined at this level typically pertains to all elements of the hierarchy throughout the company, from organizations to individual users, because corporate settings tend to be broad in scope and define boundaries for the subordinate levels to follow. In this example, there is a subordinate level in Company X **100** which represents the development organization **105** and the sales organization **110**. At this level, each subordinate organization uses the information from Company X **100** and may specify additional information to be used by its subordinate levels. Either or both of the development and sales organizations may also make changes (i.e. augmenting or overriding defined values) to the original information from the Company X level. This changed information is then made available to the subordinates of the level at which the change was made.

Moving down the hierarchy to the next level under development **105**, we have two more organizations or possibly groups (such as a department of employees): (1) the user interface group **115** and (2) the API group **120**. Both of these groups will inherit the information provided by the higher levels in this hierarchy. These groups also may add or modify information to further refine the data from the higher levels of the hierarchy, where these changes are then made available to the subordinate levels of the hierarchy. In this example, the subordinates to the user interface group **115** are the users (**140**, **145**) of the company resources. Here, for example, Sue **140** inherits all the information available from following a path from the node **140** representing Sue up through and including all of the higher levels in the hierarchy.

The information provided to Sue **140** may include values for things such as system configuration attributes, including the colors displayed on her screen and the size of the windows on her display. Sue may choose to override these values (or any values provided by a higher level) in order to customize her display to her own preferences. In doing so, Sue has now created her own customized set of values representing a coalesced version of all the higher level values from the highest level **100** down to and including the values Sue **140** provides. This customized set of information is typically stored in a database or other repository on a server for access by the user any time he needs to use the information. (It should be noted that overriding values defined at a higher level of the hierarchy may in some cases be limited to a subset of all the defined values, since some higher level values may be mandatory.)

When a user of a client machine requests data that is stored in a hierarchical structure (such as the example for FIG. 1), what the user wants in response is the "coalesced" set of data. As used herein, a coalesced set of data refers to gathering all the data in the hierarchy, and merging data together such that the value for any given attribute is set by the lowest level in the hierarchy which has that attribute defined. From the user's point of view, he receives a complete set of data, which is really a composite of the actual data from the user level and all levels above it in the hierarchical chain.

These complex types of stored data may impact overall performance of the server when information is retrieved since the coalescing of the stored data may involve a combination of several memory accesses as well as disk retrievals to collect, and then coalesce, all the necessary information before responding to a user request. For example, if Sue requests information for the hierarchy of FIG. 1, separate disk accesses (which may further require round trips through a distributed network) may be needed to retrieve the information for Company X **100**, Development **105**, User Interface **115**, and finally for Sue **140**.

While only a simple hierarchy is shown in FIG. **1**, many more levels (with many nodes at each level) may exist in an organization or corporation. Performing multiple disk accesses to retrieve information for each level is a computationally expensive operation. In addition, having to continually recalculate the coalesced image of the data when changes occur is compute intensive. Conversely, caching the complete set of coalesced data for every user is very storage or memory intensive and requires a complete recoalescence any time data represented in the coalescence is changed.

Accordingly, what is needed is a technique that avoids the performance penalty of this continual recalculation, and avoids the storage penalty of storing large amounts of coalesced data.

SUMMARY OF THE INVENTION

An object of the present invention is to provide a technique which avoids continual recalculation for coalescing complex hierarchical data and minimizes the impact on performance and memory consumption in a client/server environment.

Another object of the present invention is to provide this technique using a two-tiered cache.

Yet another object of the present invention is to provide this technique in a manner that allows a server to determine

if a set of coalesced cached values is out-of-date with the data store it represents.

Other objects and advantages of the present invention will be set forth in part in the description and in the drawings which follow and, in part, will be obvious from the description or may be learned by practice of the invention.

To achieve the foregoing objects, and in accordance with the purpose of the invention as broadly described herein, the present invention provides a method, system, and computer program product for use in a computing environment for using a two-tiered cache for storing and accessing hierarchical data. This technique comprises: providing a hierarchical structure of data comprising a top-level node, one or more intermediate levels having one or more intermediate-level nodes, and one or more user nodes, wherein each of the user nodes is a child node of the top-level node or one of the intermediate-level nodes, wherein the hierarchical structure is stored in a data repository accessible in the computing environment and wherein each of the nodes has a corresponding last update timestamp stored in the repository, the last update timestamp for the user nodes and the top-level node representing a last update to one or more data values of the node and the last update timestamp for the intermediate nodes representing an update of data values of the intermediate node or a parent of the intermediate node; creating coalesced data images (CDIs) for each of the top-level or intermediate-level nodes which is a group node, wherein a particular node is one of the group nodes when the particular node has one or more of the user nodes as a child, and wherein the CDI for the particular node comprises a coalescence of data values for the particular node, the top-level node, and all of the intermediate-level nodes in a hierarchical path from the particular node to the top-level node; storing the created CDIs in a central data cache along with a CDI timestamp for each of the stored CDIs wherein the CDI timestamp for each of the CDIs is set to the corresponding last update timestamp for the corresponding node; and storing user data for each of one or more users in a client cache for the user along with a client cache timestamp, wherein: each of the users is associated with a selected one of the user nodes; the client cache timestamp is set to the corresponding last update timestamp for the corresponding user node; and the stored user data is uncoalesced.

This technique may further comprise updating one or more of the data values for a selected node, where this updating further comprises: applying an update to the data values in the repository; updating the last update timestamp corresponding to the selected node; determining whether the selected node is one of the group nodes; and propagating the updated timestamp to each of the group nodes subordinate to the selected group node in the hierarchical structure when the determination has a positive result.

Preferably, this technique further comprises retrieving a coalesced result in response to a request from a particular one of the users, wherein the request may specify a refreshed result or an unrefreshed result.

Retrieving a refreshed result preferably further comprises: retrieving the user data for the particular user; retrieving the CDI for the group node of which the user node is one of the children; and merging the retrieved user data with the CDI of the parent node. Retrieving the user data preferably further comprises: retrieving the user data from the client cache when (1) the client cache for the particular user exists and (2) the client cache timestamp for the particular user's client cache is not older than the last update timestamp corresponding to the particular user's user node in the repository; and populating the particular user's client cache otherwise. This populating further comprises: retrieving the user data from the particular user's user node in the repository; storing the retrieved user data in the client cache for the user; and setting the client cache timestamp for the user to the last update timestamp for the corresponding user node. Retrieving the CDI for the group node preferably comprises: retrieving the CDI from the central data cache when (1) the CDI for the group node exists and (2) the CDI timestamp for the CDI is not older than the last update timestamp corresponding to the group node in the repository; and creating the CDI otherwise. Creating the CDI further comprises: retrieving the data values from the repository for the group node, the top-level node, and all of the intermediate-level nodes in the hierarchical path from the group node to the top-level node; coalescing the retrieved data values; storing the coalesced data values as the CDI for the group node in the central data cache; and setting the CDI timestamp for the CDI to the last update timestamp for the corresponding group node. This creating may further comprise repeating this process for each of the group nodes above this group node in the hierarchical path until reaching a first of the group nodes for which the CDI timestamp for the CDI of this first group node is not older than the last update timestamp corresponding to the group node in the repository.

In one aspect, the retrieving of user data from the client cache determines whether the client cache timestamp for the particular user's client cache is different from the last update timestamp corresponding to the particular user's user node in the repository rather than whether the client cache timestamp is not older than the last update timestamp, and the retrieving of the CDI from the central data cache determines whether the CDI timestamp for the CDI is different from the last update timestamp corresponding to the group node in the repository rather than whether the CDI timestamp is not older than the last update timestamp.

Retrieving an unrefreshed result preferably further comprises: retrieving the user data for the particular user; retrieving the CDI for the group node of which the user node is one of the children; and merging the retrieved user data with a parent CDI associated with a parent node of the user node. Retrieving the user data further comprises: retrieving the user data from the client cache when the client cache for the particular user exists; and populating the particular user's client cache otherwise. This populating further comprises: retrieving the user data from the particular user's user node in the repository; storing the retrieved user data in the client cache for the user; and setting the client cache timestamp for the user to the last update timestamp for the corresponding user node. Retrieving the CDI further comprises: retrieving the CDI from the central data cache when the CDI for the group node exists; and creating the CDI otherwise. Creating the CDI further comprises: retrieving the data values from the repository for the group node, the top-level node, and all of the intermediate-level nodes in the hierarchical path from the group node to the top-level node; coalescing the retrieved data values; storing the coalesced data values as the CDI for the group node in the central data cache; and setting the CDI timestamp for the CDI to the last update timestamp for the corresponding group node.

The present invention will now be described with reference to the following drawings, in which like reference numbers denote the same element throughout.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram example representing a hierarchically structured organization and a hierarchical struc-

ture for storing data associated with the organization in which the present invention may be practiced;

FIG. **2** is a block diagram of a computer workstation environment in which the present invention may be practiced;

FIG. **3** is a diagram of a networked computing environment in which the present invention may be practiced;

FIG. **4** depicts the hierarchy of FIG. **1**, showing how a two-tiered cache is used with this hierarchy, according to the present invention; and

FIGS. **5–7** provide flow charts illustrating the preferred embodiment of the logic used to implement the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. **2** illustrates a representative workstation hardware environment in which the present invention may be practiced. The environment of FIG. **2** comprises a representative single user computer workstation **210**, such as a personal computer, including related peripheral devices. The workstation **210** includes a microprocessor **212** and a bus **214** employed to connect and enable communication between the microprocessor **212** and the components of the workstation **210** in accordance with known techniques. The workstation **210** typically includes a user interface adapter **216**, which connects the microprocessor **212** via the bus **214** to one or more interface devices, such as a keyboard **218**, mouse **220**, and/or other interface devices **222**, which can be any user interface device, such as a touch sensitive screen, digitized entry pad, etc. The bus **214** also connects a display device **224**, such as an LCD screen or monitor, to the microprocessor **212** via a display adapter **226**. The bus **214** also connects the microprocessor **212** to memory **228** and long-term storage **230** which can include a hard drive, diskette drive, tape drive, etc.

The workstation **210** may communicate with other computers or networks of computers, for example via a communications channel or modem **232**. Alternatively, the workstation **210** may communicate using a wireless interface at **232**, such as a CDPD (cellular digital packet data) card. The workstation **210** may be associated with such other computers in a local area network (LAN) or a wide area network (WAN), or the workstation **210** can be a client in a client/server arrangement with another computer, etc. All of these configurations, as well as the appropriate communications hardware and software, are known in the art.

FIG. **3** illustrates a data processing network **340** in which the present invention may be practiced. The data processing network **340** may include a plurality of individual networks, such as wireless network **342** and network **344**, each of which may include a plurality of individual workstations **210**. Additionally, as those skilled in the art will appreciate, one or more LANs may be included (not shown), where a LAN may comprise a plurality of intelligent workstations coupled to a host processor.

Still referring to FIG. **3**, the networks **342** and **344** may also include mainframe computers or servers, such as a gateway computer **346** or application server **347** (which may access a data repository **348**). A gateway computer **346** serves as a point of entry into each network **344**. The gateway **346** may be preferably coupled to another network **342** by means of a communications link **350***a*. The gateway **346** may also be directly coupled to one or more workstations **210** using a communications link **350***b*, **350***c*. The gateway computer **346** may be implemented utilizing an Enterprise Systems Architecture/370 available from the International Business Machines Corporation ("IBM"), an Enterprise Systems Architecture/390 computer, etc. Depending on the application, a midrange computer, such as an Application System/400 (also known as an AS/400) may be employed. ("Enterprise Systems Architecture/370" is a trademark of IBM; "Enterprise Systems Architecture/390", "Application System/400", and "AS/400" are registered trademarks of IBM.)

The gateway computer **346** may also be coupled **349** to a storage device (such as data repository **348**). Further, the gateway **346** may be directly or indirectly coupled to one or more workstations **210**.

Those skilled in the art will appreciate that the gateway computer **346** may be located a great geographic distance from the network **342**, and similarly, the workstations **210** may be located a substantial distance from the networks **342** and **344**. For example, the network **342** may be located in California, while the gateway **346** may be located in Texas, and one or more of the workstations **210** may be located in New York. The workstations **210** may connect to the wireless network **342** using a networking protocol such as the Transmission Control Protocol/Internet Protocol ("TCP/IP") over a number of alternative connection media, such as cellular phone, radio frequency networks, satellite networks, etc. The wireless network **342** preferably connects to the gateway **346** using a network connection **350***a* such as TCP or UDP (User Datagram Protocol) over IP, X.25, Frame Relay, ISDN (Integrated Services Digital Network), PSTN (Public Switched Telephone Network), etc. The workstations **210** may alternatively connect directly to the gateway **346** using dial connections **350***b* or **350***c*. Further, the wireless network **342** and network **344** may connect to one or more other networks (not shown), in an analogous manner to that depicted in FIG. **3**.

Software programming code which embodies the present invention is typically accessed by the microprocessor **212** of the workstation **210** and server **347** from long-term storage media **230** of some type, such as a CD-ROM drive or hard drive. The software programming code may be embodied on any of a variety of known media for use with a data processing system, such as a diskette, hard drive, or CD-ROM. The code may be distributed on such media, or may be distributed to users from the memory or storage of one computer system over a network of some type to other computer systems for use by users of such other systems. Alternatively, the programming code may be embodied in the memory **228**, and accessed by the microprocessor **212** using the bus **214**. The techniques and methods for embodying software programming code in memory, on physical media, and/or distributing software code via networks are well known and will not be further discussed herein.

A user of the present invention at a client computer may connect his computer to a server using a wireline connection, or a wireless connection. Wireline connections are those that use physical media such as cables and telephone lines, whereas wireless connections use media such as satellite links, radio frequency waves, and infrared waves. Many connection techniques can be used with these various media, such as: using the computer's modem to establish a connection over a telephone line; using a LAN card such as Token Ring or Ethernet; using a cellular modem to establish a wireless connection; etc. The user's computer may be any type of computer processor, including laptop, handheld or mobile computers; vehicle-mounted devices; desktop computers; mainframe computers; etc., having processing and communication capabilities. The remote server, similarly,

can be one of any number of different types of computer which have processing and communication capabilities. These techniques are well known in the art, and the hardware devices and software which enable their use are readily available. Hereinafter, the user's computer will be referred to equivalently as a "workstation", "device", or "computer", and use of any of these terms or the term "server" refers to any of the types of computing devices described above.

In the preferred embodiment, the present invention is implemented as one or more modules (also referred to as code subroutines, or "objects" in object-oriented programming) of a computer software program (or programs) which provides a two-tiered caching technique to avoid performance impacts and to avoid storing large amounts of data in memory. The invention may be used in the Internet environment or in a corporate intranet, extranet, or any other network environment. The program code of the preferred embodiment may be implemented as objects in an object-oriented programming language, or in a conventional procedurally-oriented language, or in a mix of object-oriented and procedural language code. In the preferred embodiment, the code of the present invention operates on the server. Alternatively, a portion of the code may operate on the client machine (e.g. to perform the coalescing of a user's preferences with data inherited from higher levels).

A server machine in the client/server environment typically maintains frequently accessed information in a system memory cache for improved performance. For complex data structures, such as a hierarchy of data as has been discussed, the cached image may be comprised of the coalesced data from all levels of the hierarchy. The problem with maintaining a cached image for a full hierarchy of data is that the image must be recalculated (i.e. re-coalesced) each time a change is made to the information which is represented by the cached image, in order to incorporate the changes into the image. With reference to the example hierarchy of FIG. **1**, a coalescence of the full hierarchy (according to the prior art) would exist for each user, to include unique information for this user. The coalescing of data for each user may result in multiple cached images that are essentially the same, except for the individual user's unique values. In addition, coalesced images may be stored for each intermediate level (representing the hierarchical path to the top of the structure). System performance is affected by all the required frequent coalescing of data and the memory utilization is affected since multiple versions of the hierarchical data are stored in the system memory cache.

According to the preferred embodiment of the present invention, this problem is solved by keeping two levels of cache: (1) a central cache of coalesced images, and (2) a client cache for uncoalesced client images. In the preferred embodiment, these two caches exist in the same physical memory cache on the server machine and are logically separated. Optionally, these two caches may be physically separate memory caches on the same server machine or may be located on physically separate machines (including storing a client's uncoalesced data locally on the client machine).

According to the present invention, a coalesced data image (CDI) of hierarchical data is kept for all levels of the hierarchy which have users (or more generally, terminating resources) beneath them. Each user who provides customized data to override the inherited data in a CDI has his unique changes saved in a client cache (where the changes are stored uncoalesced). (A user's unique changes are referred to herein as the user's "uncoalesced data".) When the user requests the coalesced data for all the information represented by the hierarchy, the CDI of the user's immediate parent (i.e. the next higher level of the hierarchy) is retrieved from the central cache and is coalesced with the uncoalesced image in the client cache. The result is a single, coalesced image comprised of all the values provided by the hierarchy as well as the unique values specified by the user which override or augment values inherited from the hierarchy. In this manner, the volatility of the central cache is significantly reduced since each CDI does not include the more volatile client information. The amount of memory is significantly reduced since multiple users (such as Sue **140** and Bob **145**) can share the same CDI. By caching a coalesced image representing all levels of a hierarchy above a particular user (such as CDI **400** for the Development level **105** above user Bob **135**), or multiple users (such as CDI **405** for the User Interface level **115** above user Sue **140** and Tim **145**), and then applying a user's uncoalesced data to the appropriate CDI, the user obtains a full set of information with only two storage retrievals (one for the CDI, and one for the user's uncoalesced data)—assuming the CDI is up-to-date.

Note that while the hierarchy discussed herein is directed toward configuration data in an organization, this is merely illustrative. A typical data store may contain hierarchically structured information about various types of resources. (Examples of a resource could be a user, group, machine, etc.) From the user's point of view, these resources are related logically in a hierarchical group structure. So, in the general case, a resource might be a member of a group, and that group might be a member of a higher level group, etc.

The preferred embodiment of the present invention will now be discussed with regard to FIGS. **4–7**.

FIG. **4** shows how the two-tiered cache of the present invention is used with hierarchically stored data such as the example depicted in FIG. **1**. As discussed earlier, a coalesced data image of hierarchical data is kept for all levels of the hierarchy which have users beneath them. Thus, in FIG. **4**, a coalesced data image **400, 405,** is kept in the central cache for the "Development" **105** and "User Interface" **115** nodes because these groups have users Bob **135**, and Sue **140** and Tim **145**, respectively. A client cache is kept for each user (Bob **135**, Sue **140**, Tim **145**), and this client cache provides uncoalesced data (**410, 415, 420**, respectively) to override the data stored in the CDI for the user's immediate parent. The other nodes of the example hierarchy have no users depicted, and thus none of these nodes has an associated CDI. The coalesced data image **400** for the Development group **105** is comprised of all information from the Development group and all levels above it (i.e. the Company **100**). The coalesced data image **405** for the User Interface group **115** is comprised of all the information from the User Interface group **115**, the Development group **105**, and the Company **100**. When user Sue **140**, for example, requests the coalesced data from the hierarchy, the CDI **405** from her immediate parent (User Interface **115**) is coalesced with the uncoalesced data **415** from Sue's client cache and returned to Sue **140**.

The coalesced data in a CDI typically changes infrequently since it contains data provided at higher, less volatile levels of a hierarchy. That is, it can be expected that a department changes its information less frequently than does a user, that a division changes its information less frequently than a department, etc.

According to the present invention, CDIs stored in the central cache are read-only in that changes are never made by users directly in the data represented by a CDI. At some

point in time, an administrator may have a need to change some or all of the data in the hierarchy. (Administrators typically have authority to make changes to data at various levels of the hierarchy, and do not have the same limitations as other users.) Should an administrator change higher level data (i.e. non-user-level data such as corporate-level data, division-level data, etc.) that is stored on the server, the CDI will be out-of-date with the stored data. By requesting the hierarchical data with the refresh option (discussed in detail below), the present invention will cause the server to re-coalesce all the data into a new coalesced data image in the central cache. Note that multiple new CDIs may be created during a refresh, when the changed data is represented in more than one CDI. For example, information defined at Company level **100** is reflected in CDI **400** as well as CDI **405**. Similarly, information defined at Development level **105** is reflected in both of these CDIs. Thus, if an administrator changes Company **100** data or Development **105** data, both CDIs will become out-of-date and will be re-coalesced on a refresh issued from a level beneath User Interface **115**.

The logic with which the preferred embodiment of the present invention is implemented will now be discussed with reference to FIGS. **5–7**.

FIG. **5** provides a flow chart illustrating the logic with which a user requests coalesced data from a server without requesting a refresh. In this example, the user "Joe" at Block **500** issues a request to a server for data that is coalesced from complex, hierarchical data. The desired coalesced data in this example is comprised of all the data stored in the hierarchy including any data previously specified by Joe (e.g. additional values or overridden values). At Block **505**, a determination is made as to whether a client cache exists for this user Joe. If there is no client cache (a "No" at Block **505**), then at Block **510** the client's uncoalesced data is retrieved from the data store (i.e. disk file, database, etc.) and used to populate a client cache entry for this user. A timestamp representing the last update to this user's uncoalesced data in the repository is also stored-in the user's client cache. If, at Block **505**, a client cache for this user does exist (a "Yes" result), then processing continues to Block **515**.

At Block **515**, a test is made to determine if a CDI for the parent (i.e the next higher level above the requesting user) exists in the central cache. If not, then processing continues to Block **520** where all the data for the higher-level groups in the hierarchy is gathered, from the current user's parent level to the highest level. The collected information is then coalesced and stored as a CDI in the central cache. A timestamp associated with this CDI is saved in the central cache at Block **525**, where the value of this timestamp is set to the last update time of the data at this level in the repository.

If a CDI for the parent group exists (a "Yes" result at Block **515**), or after creating a CDI at Blocks **520** and **525**, processing continues at Block **530**. At Block **530**, the user's uncoalesced data in the client cache is coalesced with the CDI for the user's parent, and the result is returned to the user. Note that the logic of FIG. **5** made no determination as to whether values in an existing CDI were out-of-date with the corresponding data stored in the data repository since the user's request (Block **500**) did not request a refresh of any previously cached values.

FIG. **6** provides a flow chart illustrating the preferred embodiment of the logic for processing a user's request for coalesced data which will be refreshed (if the appropriate CDI and/or client cache is out-of-date). The user (at Block **600**) issues a request for the coalesced preferences (as before), but now requests a refresh of any out-of-date cached values. Block **605** determines if the client cache for this user is up-to-date. A client cache for the requesting user may not exist when the request for a refresh is received. In that case, a client cache is built which, by default, is refreshed. Otherwise, a comparison is made between the timestamp of this user's client cache and the timestamp indicating the last update to the user's stored data in the data repository. If the existing client cache for this user is not up-to-date (i.e. the client cache timestamp is older than the timestamp of the last update to the user's stored data), then at Block **610**, any existing client cache data is discarded. The user's client cache is then populated with current, uncoalesced data, and the timestamp indicating the last time the user's stored data was updated is saved in the client cache. If the user's client cache exists and is not out-of-date, processing continues to Block **615**.

Block **615** determines if the CDI for the parent group exists in the central cache and is up-to-date. As with the client cache, if the CDI for the parent group does not exist, then a new parent CDI is created in the central cache at Block **620** which, by default, is refreshed. Or, if the CDI for the parent group is not up-to-date (i.e. the CDI timestamp for the parent group is older than the timestamp of the last update at this level in the repository), then Block **620** discards any existing CDI for the parent group and gathers all the data for the higher-level groups in the hierarchy from the repository, including the parent group. The collected data is coalesced into a (new or refreshed) CDI, and the central cache is populated with this CDI. Block **625** saves the timestamp associated with this CDI in the central cache, where the CDI timestamp is the time of the last update to data for this level or for one of the higher-level groups coalesced into this CDI, whichever is later, in the repository. Optionally, this checking of the parent group CDI may be repeated for each next-higher-level group CDI in the path to the top level, until reaching a CDI which is up-to-date.

Block **630** is reached following a positive result at Block **615** or completion of Block **625**. At Block **630**, the CDI of the user's parent group is coalesced with the uncoalesced data in the user's client cache, and the result is returned to the user.

FIG. **7** provides a flowchart illustrating the logic with which the preferred embodiment processes a user request to update data stored in the data repository. At Block **700**, the user issues a request to update data which is stored in a repository. At Block **705**, a determination is made as to whether the data being updated exists in the client cache for this user. If not, then at Block **710** the user's client cache is populated with the existing uncoalesced data and the timestamp of the last update to the user's stored data is saved in the client cache. The user's specified updates are made to the stored data as well as to the client cache at Block **715**. At Block **720**, the timestamp of the client's data in the repository as well as the client cache timestamp are updated to reflect the time of the current update. Block **725** determines if the updated information is group-level data. If so, then an updated timestamp is propagated downward (Block **730**) to all subordinate groups of the hierarchy below this updated group. That is, the last update time for the subordinate groups in the repository is set to the time of the current update. By changing the timestamps of the stored data for the group and its subordinate groups, a refresh of any CDIs for these subordinate data levels will occur with the nekt user request that specifies a refresh. Block **735** returns an indication of success or failure of the update operation to the user.

In an optional enhancement of the preferred embodiment, a randomly-generated sequence number may be used as the "timestamp" values which have been discussed herein, instead of a conventional time value. This enhancement will be advantageous when more than one server application may be updating data values in the repository (and thus updating the associated timestamps), and enables avoiding clock synchronization discrepancies among the different applications. When this enhancement is used, the comparisons between timestamp values to determine if a stored CDI or client cache is up-to-date is replaced by a comparison to determine whether the timestamp values are different: if the values are different, then the stored data has been changed and thus is not up-to-date.

As has been demonstrated, the present invention provides advantageous techniques for storing and retrieving hierarchically structured data by creating a two-tiered cache. The technique defined herein for creating and accessing this two-tiered cache avoids the performance penalty and storage penalty of prior art solutions.

While the preferred embodiment of the present invention has been described, additional variations and modifications in that embodiment may occur to those skilled in the art once they learn of the basic inventive concepts. Therefore, it is intended that the appended claims shall be construed to include both the preferred embodiment and all such variations and modifications as fall within the spirit and scope of the invention.

We claim:

1. A computer program product embodied on a computer-readable medium in a computing environment, for using a two-tiered cache for storing and accessing hierarchical data, comprising:

a hierarchical structure of data comprising a top-level node, one or more intermediate levels having one or more intermediate-level nodes, and one or more user nodes, wherein each of said user nodes is a child node of said top-level node or one of said intermediate-level nodes, wherein said hierarchical structure is stored in a data repository accessible in said computing environment and wherein each of said nodes has a corresponding last update timestamp stored in said repository, said last update timestamp for said user nodes and said top-level node representing a last update to one or more data values of said node and said last update timestamp for said intermediate nodes representing an update of data values of said intermediate node or a parent of said intermediate node;

computer-readable program code means for creating coalesced data images (CDIs) for each of said top-level or intermediate-level nodes which is a group node, wherein a particular node is one of said group nodes when said particular node has one or more of said user nodes as a child, and wherein said CDI for said particular node comprises a coalescence of data values for said particular node, said top-level node, and all of said intermediate-level nodes in a hierarchical path from said particular node to said top-level node;

computer-readable program code means for storing said created CDIs in a central data cache along with a CDI timestamp for each of said stored CDIs wherein said CDI timestamp for each of said CDIs is set to said corresponding last update timestamp for said corresponding node; and

computer-readable program code means for storing user data for each of one or more users in a client cache for said user along with a client cache timestamp, wherein: each of said users is associated with a selected one of said user nodes; said client cache timestamp is set to said corresponding last update timestamp for said corresponding user node; and said stored user data is uncoalesced.

2. The computer program product according to claim 1, further comprising:

computer readable program code means for updating one or more of said data values for a selected node, further comprising:

computer readable program code means for applying an update to said data values in said repository;

computer readable program code means for updating said last update timestamp corresponding to said selected node;

computer readable program code means for determining whether said selected node is one of said group nodes; and

computer readable program code means for propagating said updated timestamp to each of said group nodes subordinate to said selected group node in said hierarchical structure when said computer readable program code means for determining has a positive result.

3. The computer program product according to claim 2, further comprising:

computer readable program code means for retrieving a coalesced result in response to a request from a particular one of said users, wherein said request may specify a refreshed result or an unrefreshed result.

4. The computer program product according to claim 3, wherein said computer readable program code means for retrieving a refreshed result further comprises:

computer readable program code means for retrieving said user data for said particular user, further comprising:

computer readable program code means for retrieving said user data from said client cache when (1) said client cache for said particular user exists and (2) said client cache timestamp for said particular user's client cache is not older than said last update timestamp corresponding to said particular user's user node in said repository; and

computer readable program code means for populating said particular user's client cache otherwise, wherein said populating further comprises:

computer readable program code means for retrieving said user data from said particular user's user node in said repository;

computer readable program code means for storing said retrieved user data in said client cache for said user; and

computer readable program code means for setting said client cache timestamp for said user to said last update timestamp for said corresponding user node; and

computer readable program code means for retrieving said CDI for said group node of which said user node is one of said children, further comprising:

computer readable program code means for retrieving said CDI from said central data cache when (1) said CDI for said group node exists and (2) said CDI timestamp for said CDI is not older than said last update timestamp corresponding to said group node in said repository; and

computer readable program code means for creating said CDI otherwise, wherein said creating further comprises:

computer readable program code means for retrieving said data values from said repository for said group node, said top-level node, and all of said intermediate-level nodes in said hierarchical path from said group node to said top-level node;

computer readable program code means for coalescing said retrieved data values;

computer readable program code means for storing said coalesced data values as said CDI for said group node in said central data cache; and

computer readable program code means for setting said CDI timestamp for said CDI to said last update timestamp for said corresponding group node; and

computer readable program code means for merging said retrieved user data with said retrieved CDI.

5. The computer program product according to claim 3, wherein said computer readable program code means for retrieving an unrefreshed result further comprises:

computer readable program code means for retrieving said user data for said particular user, further comprising:

computer readable program code means for retrieving said user data from said client cache when said client cache for said particular user exists; and

computer readable program code means for populating said particular user's client cache otherwise, wherein said populating further comprises:

computer readable program code means for retrieving said user data from said particular user's user node in said repository;

computer readable program code means for storing said retrieved user data in said client cache for said user; and

computer readable program code means for setting said client cache timestamp for said user to said last update timestamp for said corresponding user node;

computer readable program code means for retrieving said CDI for said group node of which said user node is one of said children, further comprising:

computer readable program code means for retrieving said CDI from said central data cache when said CDI for said group node exists; and

computer readable program code means for creating said CDI otherwise, wherein said creating further comprises:

computer readable program code means for retrieving said data values from said repository for said group node, said top-level node, and all of said intermediate-level nodes in said hierarchical path from said group node to said top-level node;

computer readable program code means for coalescing said retrieved data values;

computer readable program code means for storing said coalesced data values as said CDI for said group node in said central data cache; and

computer readable program code means for setting said CDI timestamp for said CDI to said last update timestamp for said corresponding group node; and

computer readable program code means for merging said retrieved user data with a parent CDI associated with a parent node of said user node.

6. The computer program product according to claim 4, wherein said computer readable program code means for creating further comprises:

computer readable program code means for repeating said creating for each of said group nodes above said group node in said hierarchical path until reaching a first of said group nodes for which said CDI timestamp for said CDI of said first group node is not older than said last update timestamp corresponding to said group node in said repository.

7. The computer program product according to claim 4, wherein:

said computer readable program code means for retrieving said user data from said client cache determines whether said client cache timestamp for said particular user's client cache is different from said last update timestamp corresponding to said particular user's user node in said repository rather than whether said client cache timestamp is not older than said last update timestamp; and

said computer readable program code means for retrieving said CDI from said central data cache determines whether said CDI timestamp for said CDI is different from said last update timestamp corresponding to said group node in said repository rather than whether said CDI timestamp is not older than said last update timestamp.

8. A system for using a two-tiered cache for storing and accessing hierarchical data, comprising:

a hierarchical structure of data comprising a top-level node, one or more intermediate levels having one or more intermediate-level nodes, and one or more user nodes, wherein each of said user nodes is a child node of said top-level node or one of said intermediate-level nodes, wherein said hierarchical structure is stored in a data repository accessible in said computing environment and wherein each of said nodes has a corresponding last update timestamp stored in said repository, said last update timestamp for said user nodes and said top-level node representing a last update to one or more data values of said node and said last update timestamp for said intermediate nodes representing an update of data values of said intermediate node or a parent of said intermediate node;

means for creating coalesced data images (CDIs) for each of said top-level or intermediate-level nodes which is a group node, wherein a particular node is one of said group nodes when said particular node has one or more of said user nodes as a child, and wherein said CDI for said particular node comprises a coalescence of data values for said particular node, said top-level node, and all of said intermediate-level nodes in a hierarchical path from said particular node to said top-level node;

means for storing said created CDIs in a central data cache along with a CDI timestamp for each of said stored CDIs wherein said CDI timestamp for each of said CDIs is set to said corresponding last update timestamp for said corresponding node; and

means for storing user data for each of one or more users in a client cache for said user along with a client cache timestamp, wherein: each of said users is associated with a selected one of said user nodes; said client cache timestamp is set to said corresponding last update timestamp for said corresponding user node; and said stored user data is uncoalesced.

9. The system according to claim 8, further comprising:

means for updating one or more of said data values for a selected node, further comprising:

means for applying an update to said data values in said repository;

means for updating said last update timestamp corresponding to said selected node;

means for determining whether said selected node is one of said group nodes; and

means for propagating said updated timestamp to each of said group nodes subordinate to said selected group node in said hierarchical structure when said means for determining has a positive result.

10. The system according to claim 9, further comprising:

means for retrieving a coalesced result in response to a request from a particular one of said users, wherein said request may specify a refreshed result or an unrefreshed result.

11. The system according to claim 10, wherein said means for retrieving a refreshed result further comprises:

means for retrieving said user data for said particular user, further comprising:

means for retrieving said user data from said client cache when (1) said client cache for said particular user exists and (2) said client cache timestamp for said particular user's client cache is not older than said last update timestamp corresponding to said particular user's user node in said repository; and

means for populating said particular user's client cache otherwise, wherein said populating further comprises:

means for retrieving said user data from said particular user's user node in said repository;

means for storing said retrieved user data in said client cache for said user; and

means for setting said client cache timestamp for said user to said last update timestamp for said corresponding user node;

means for retrieving said CDI for said group node of which said user node is one of said children, further comprising:

means for retrieving said CDI from said central data cache when (1) said CDI for said group node exists and (2) said CDI timestamp for said CDI is not older than said last update timestamp corresponding to said group node in said repository; and

means for creating said CDI otherwise, wherein said creating further comprises:

means for retrieving said data values from said repository for said group node, said top-level node, and all of said intermediate-level nodes in said hierarchical path from said group node to said top-level node;

means for coalescing said retrieved data values;

means for storing said coalesced data values as said CDI for said group node in said central data cache; and

means for setting said CDI timestamp for said CDI to said last update timestamp for said corresponding group node; and

means for merging said retrieved user data with said CDI of said parent node.

12. The system according to claim 10, wherein said means for retrieving an unrefreshed result further comprises:

means for retrieving said user data for said particular user, further comprising:

means for retrieving said user data from said client cache when said client cache for said particular user exists; and

means for populating said particular user's client cache otherwise, wherein said populating further comprises:

means for retrieving said user data from said particular user's user node in said repository;

means for storing said retrieved user data in said client cache for said user; and

means for setting said client cache timestamp for said user to said last update timestamp for said corresponding user node;

means for retrieving said CDI for said group node of which said user node is one of said children, further comprising:

means for retrieving said CDI from said central data cache when said CDI for said group node exists; and

means for creating said CDI otherwise, wherein said creating further comprises:

means for retrieving said data values from said repository for said group node, said top-level node, and all of said intermediate-level nodes in said hierarchical path from said group node to said top-level node;

means for coalescing said retrieved data values;

means for storing said coalesced data values as said CDI for said group node in said central data cache; and

means for setting said CDI timestamp for said CDI to said last update timestamp for said corresponding group node; and

means for merging said retrieved user data with a parent CDI associated with a parent node of said user node.

13. The system according to claim 11, wherein said means for creating further comprises:

means for repeating said creating for each of said group nodes above said group node in said hierarchical path until reaching a first of said group nodes for which said CDI timestamp for said CDI of said first group node is not older than said last update timestamp corresponding to said group node in said repository.

14. The system according to claim 11, wherein:

said means for retrieving said user data from said client cache determines whether said client cache timestamp for said particular user's client cache is different from said last update timestamp corresponding to said particular user's user node in said repository rather than whether said client cache timestamp is not older than said last update timestamp; and

said means for retrieving said CDI from said central data cache determines whether said CDI timestamp for said CDI is different from said last update timestamp corresponding to said group node in said repository rather than whether said CDI timestamp is not older than said last update timestamp.

15. A method for using a two-tiered cache for storing and accessing hierarchical data, comprising the steps of:

providing a hierarchical structure of data comprising a top-level node, one or more intermediate levels having one or more intermediate-level nodes, and one or more user nodes, wherein each of said user nodes is a child node of said top-level node or one of said intermediate-level nodes, wherein said hierarchical structure is stored in a data repository accessible in said computing environment and wherein each of said nodes has a corresponding last update timestamp stored in said repository, said last update timestamp for said user nodes and said top-level node representing a last update to one or more data values of said node and said last update timestamp for said intermediate nodes representing an update of data values of said intermediate node or a parent of said intermediate node;

creating coalesced data images (CDIs) for each of said top-level or intermediate-level nodes which is a group node, wherein a particular node is one of said group nodes when said particular node has one or more of said user nodes as a child, and wherein said CDI for said particular node comprises a coalescence of data values for said particular node, said top-level node, and all of said intermediate-level nodes in a hierarchical path from said particular node to said top-level node;

storing said created CDIs in a central data cache along with a CDI timestamp for each of said stored CDIs wherein said CDI timestamp for each of said CDIs is set to said corresponding last update timestamp for said corresponding node; and

storing user data for each of one or more users in a client cache for said user along with a client cache timestamp, wherein: each of said users is associated with a selected one of said user nodes; said client cache timestamp is set to said corresponding last update timestamp for said corresponding user node; and said stored user data is uncoalesced.

16. The method according to claim **15**, further comprising the step of:

updating one or more of said data values for a selected node, further comprising the steps of:

applying an update to said data values in said repository;

updating said last update timestamp corresponding to said selected node;

determining whether said selected node is one of said group nodes; and

propagating said updated timestamp to each of said group nodes subordinate to said selected group node in said hierarchical structure when said determining step has a positive result.

17. The method according to claim **16**, further comprising the step of:

retrieving a coalesced result in response to a request from a particular one of said users, wherein said request may specify a refreshed result or an unrefreshed result.

18. The method according to claim **17**, wherein said retrieving a refreshed result step further comprises the steps of:

retrieving said user data for said particular user, further comprising the steps of:

retrieving said user data from said client cache when (1) said client cache for said particular user exists and (2) said client cache timestamp for said particular user's client cache is not older than said last update timestamp corresponding to said particular user's user node in said repository; and

populating said particular user's client cache otherwise, wherein said populating further comprises the steps of:

retrieving said user data from said particular user's user node in said repository;

storing said retrieved user data in said client cache for said user; and

setting said client cache timestamp for said user to said last update timestamp for said corresponding user node;

retrieving said CDI for said group node of which said user node is one of said children, further comprising the steps of:

retrieving said CDI from said central data cache when (1) said CDI for said group node exists and (2) said CDI timestamp for said CDI is not older than said last update timestamp corresponding to said group node in said repository; and

creating said CDI otherwise, wherein said creating further comprises the steps of:

retrieving said data values from said repository for said group node, said top-level node, and all of said intermediate-level nodes in said hierarchical path from said group node to said top-level node;

coalescing said retrieved data values;

storing said coalesced data values as said CDI for said group node in said central data cache; and

setting said CDI timestamp for said CDI to said last update timestamp for said corresponding group node; and

merging said retrieved user data with said CDI of said parent node.

19. The method according to claim **17**, wherein said retrieving an unrefreshed result step further comprises the steps of:

retrieving said user data for said particular user, further comprising the steps of:

retrieving said user data from said client cache when said client cache for said particular user exists; and

populating said particular user's client cache otherwise, wherein said populating step further comprises the steps of:

retrieving said user data from said particular user's user node in said repository;

storing said retrieved user data in said client cache for said user; and

setting said client cache timestamp for said user to said last update timestamp for said corresponding user node;

retrieving said CDI for said group node of which said user node is one of said children, further comprising the steps of:

retrieving said CDI from said central data cache when said CDI for said group node exists; and

creating said CDI otherwise, wherein said creating step further comprises the steps of:

retrieving said data values from said repository for said group node, said top-level node, and all of said intermediate-level nodes in said hierarchical path from said group node to said top-level node;

coalescing said retrieved data values;

storing said coalesced data values as said CDI for said group node in said central data cache; and

setting said CDI timestamp for said CDI to said last update timestamp for said corresponding group node; and

merging said retrieved user data with a parent CDI associated with a parent node of said user node.

20. The method according to claim **18**, wherein said creating step further comprises the step of:

repeating said creating step for each of said group nodes above said group node in said hierarchical path until reaching a first of said group nodes for which said CDI timestamp for said CDI of said first group node is not older than said last update timestamp corresponding to said group node in said repository.

21. The method according to claim **18**, wherein:

said retrieving said user data from said client cache step determines whether said client cache timestamp for said particular user's client cache is different from said last update timestamp corresponding to said particular user's user node in said repository rather than whether

said client cache timestamp is not older than said last update timestamp; and

said retrieving said CDI from said central data cache step determines whether said CDI timestamp for said CDI is different from said last update timestamp corresponding to said group node in said repository rather than whether said CDI timestamp is not older than said last update timestamp.

* * * * *

US005572643A

United States Patent [19]

Judson

[11] Patent Number: 5,572,643

[45] Date of Patent: Nov. 5, 1996

[54] **WEB BROWSER WITH DYNAMIC DISPLAY OF INFORMATION OBJECTS DURING LINKING**

[76] Inventor: **David H. Judson**, 6823 Northport, Dallas, Tex. 75230

[21] Appl. No.: **543,876**

[22] Filed: **Oct. 19, 1995**

[51] **Int. Cl.**[6] **G06F 19/00**

[52] **U.S. Cl.** **395/793**

[58] **Field of Search** 395/155–161, 395/145–149; 380/4

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,782,463	11/1988	Sanders et al.	395/700
4,827,508	5/1989	Shear	380/4
4,833,308	5/1989	Humble	235/383
4,953,209	8/1990	Ryder, Sr. et al.	380/23
5,204,947	4/1993	Bernstein et al.	395/157
5,297,249	3/1994	Bernstein et al.	395/156
5,355,472	10/1994	Lewis	395/600
5,359,708	10/1994	Bloomer et al.	395/148
5,367,621	11/1994	Cohen et al.	395/154
5,367,623	11/1994	Iwai et al.	395/157
5,408,659	4/1995	Cavendish et al.	395/159 X
5,412,772	5/1995	Monson	395/161 X
5,428,529	6/1995	Hatrick et al.	395/145 X
5,438,508	8/1995	Wyman	380/4 X
5,442,771	8/1995	Flepp et al.	395/650
5,461,667	10/1995	Remillard	379/96
5,491,820	2/1996	Belove et al.	395/600
5,511,160	4/1996	Robson	395/162
5,515,490	5/1996	Buchanan et al.	395/154

OTHER PUBLICATIONS

Pike et al., Using Mosaic, 1994, pp. 82–85, 222–223.

Baker, Hypertext Browsing on the Internet, UNIX Review, v. 12, No. 9, Sep. 1994, pp. 21–26.

DeVoney, Using PCDOS, 1986, p. 340, 1986.

SPRY, AIRMOS.HLP Windows Help File, Apr. 3, 1995, Browsing With Mosaic, The SPRY Mosaic Console.

Gunn, Power in Pictures, Computer Shopper, Nov. 1994, vol. 14 No. 11, pp. 598–600.

Michalski, Content in Context, RELease 1.0, vol. 94, No. 9, Sep. 27, 1994, pp. 1–13.

McArthur, World Wide Web & HTML, Dr. Dobb's Journal, Dec. 1994.

Davison, Coding With HTML Forms, Dr. Dobb's Journal, Jun. 1995, pp. 70–75, 106–109.

Grobe, Michael, "HTML Quick Reference" Oct. 11, 1995, Academic Computing Services, The University of Kansas.

Ayre, Rick and Don Willmott, "See the Sites Beyond Browsing" Oct. 10, 1995, PC Magazine, pp. 151–201.

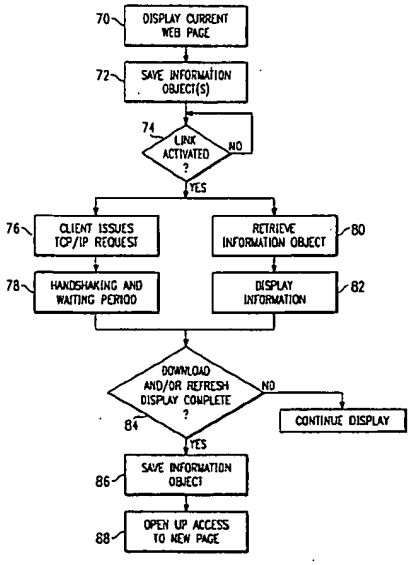
Primary Examiner—Mark K. Zimmerman

Assistant Examiner—Anton W. Fetting

[57] **ABSTRACT**

A method of browsing the Worldwide Web of the Internet using an HTML-compliant client supporting a graphical user interface and a browser. The method begins as a web page is being displayed on the graphical user interface, the web page having at least one link to a hypertext document preferably located at a remote server. In response to the user clicking on the link, the link is activated by the browser to thereby request downloading of the hypertext document from the remote server to the graphical user interface of the client. While the client waits for a reply and/or as the hypertext document is being downloaded, the browser displays one or more different types of informational messages to the user. Such messages include, for example, advertisements, notices, messages, copyright information and the like.

19 Claims, 8 Drawing Sheets



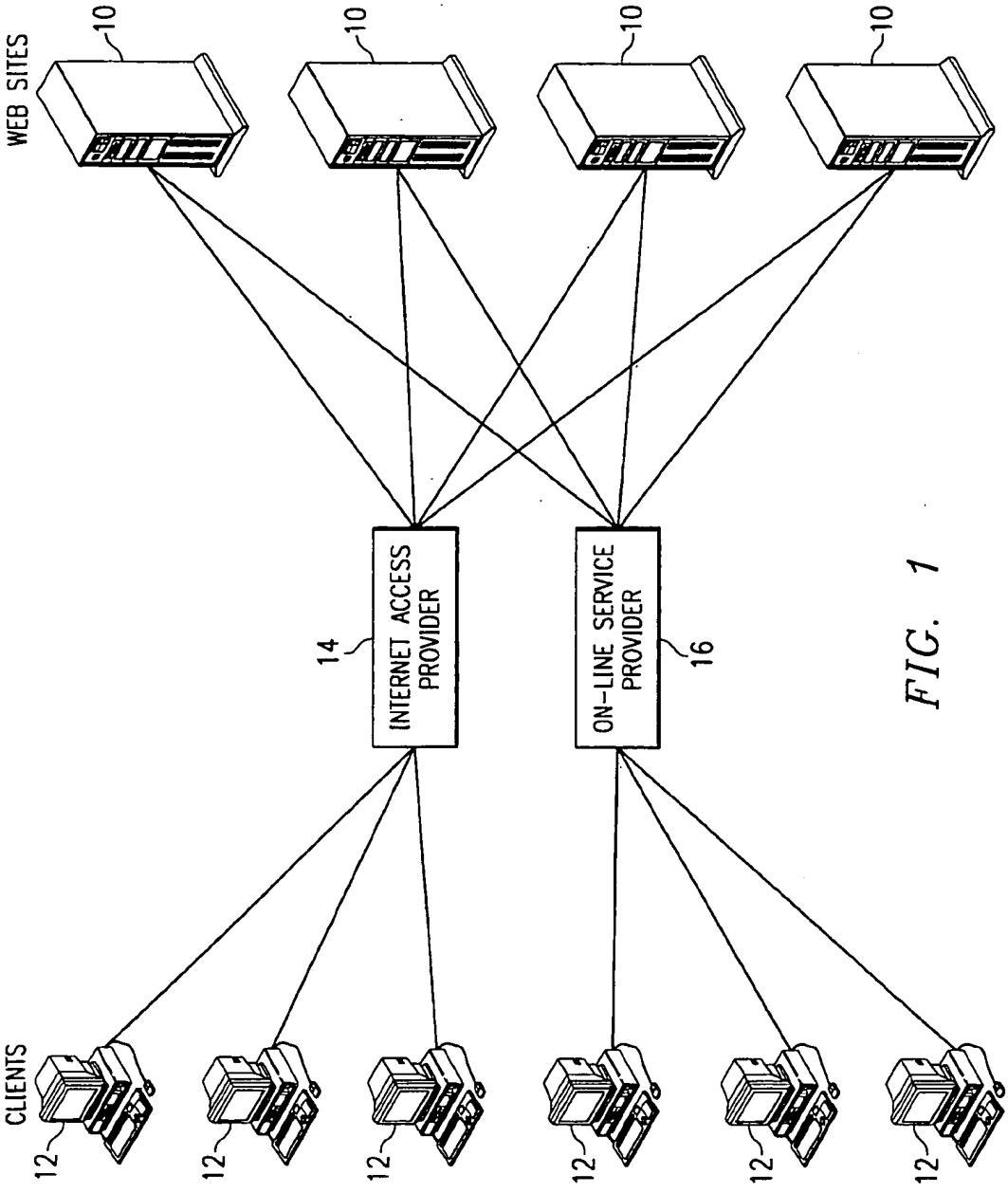
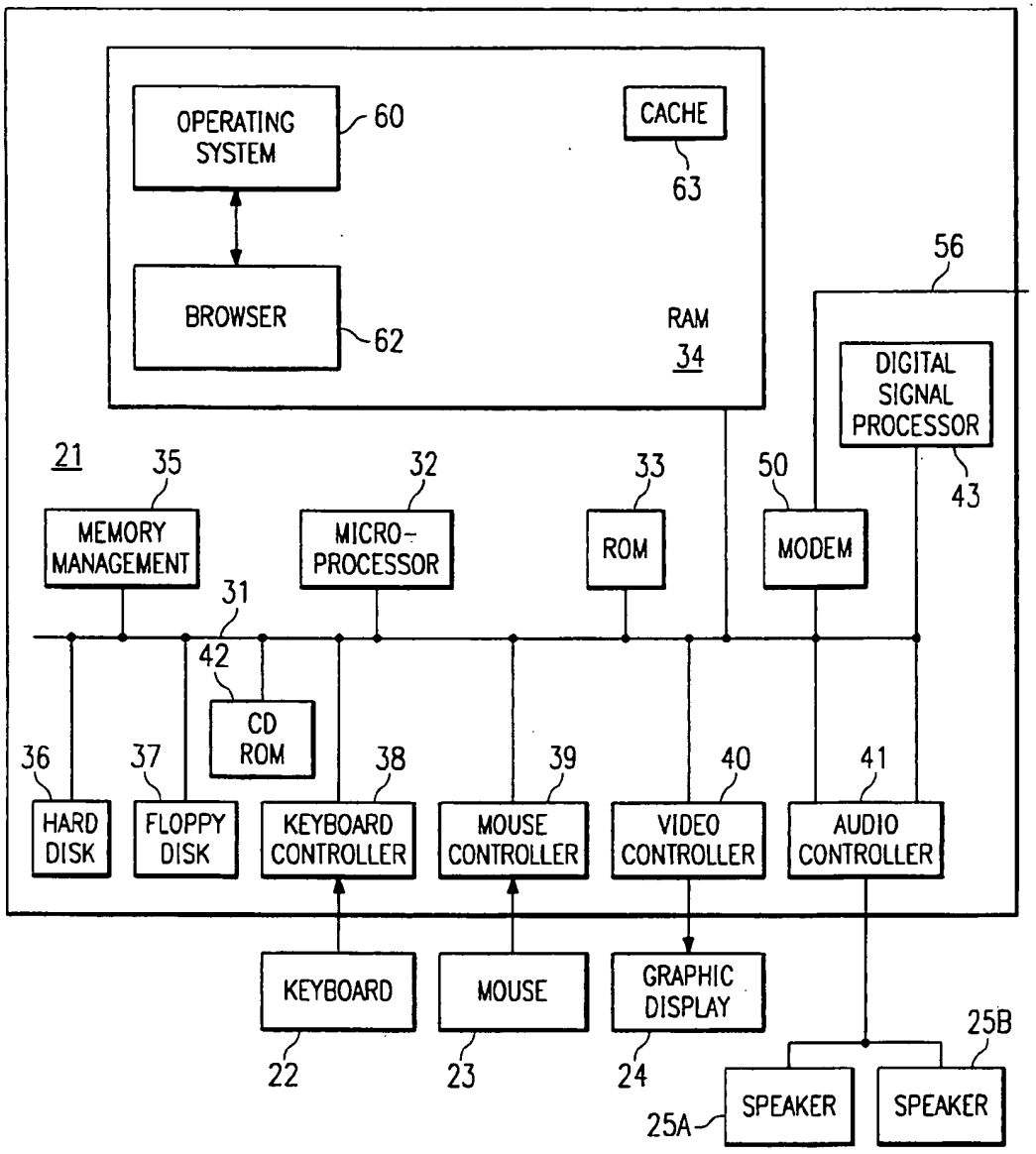

WEB SITES
10
10
10
10
INTERNET ACCESS PROVIDER
14
ON-LINE SERVICE PROVIDER
16
CLIENTS
12
12
12
12
12
12

FIG. 1

FIG. 2


OPERATING SYSTEM
60
CACHE
63
BROWSER
62
RAM
34
56
DIGITAL SIGNAL PROCESSOR
43
21
35
MEMORY MANAGEMENT
32
MICRO-PROCESSOR
33
ROM
50
MODEM
31
42
CD ROM
36
37
38
39
40
41
HARD DISK
FLOPPY DISK
KEYBOARD CONTROLLER
MOUSE CONTROLLER
VIDEO CONTROLLER
AUDIO CONTROLLER
KEYBOARD
MOUSE
GRAPHIC DISPLAY
25B
22
23
24
25A
SPEAKER
SPEAKER

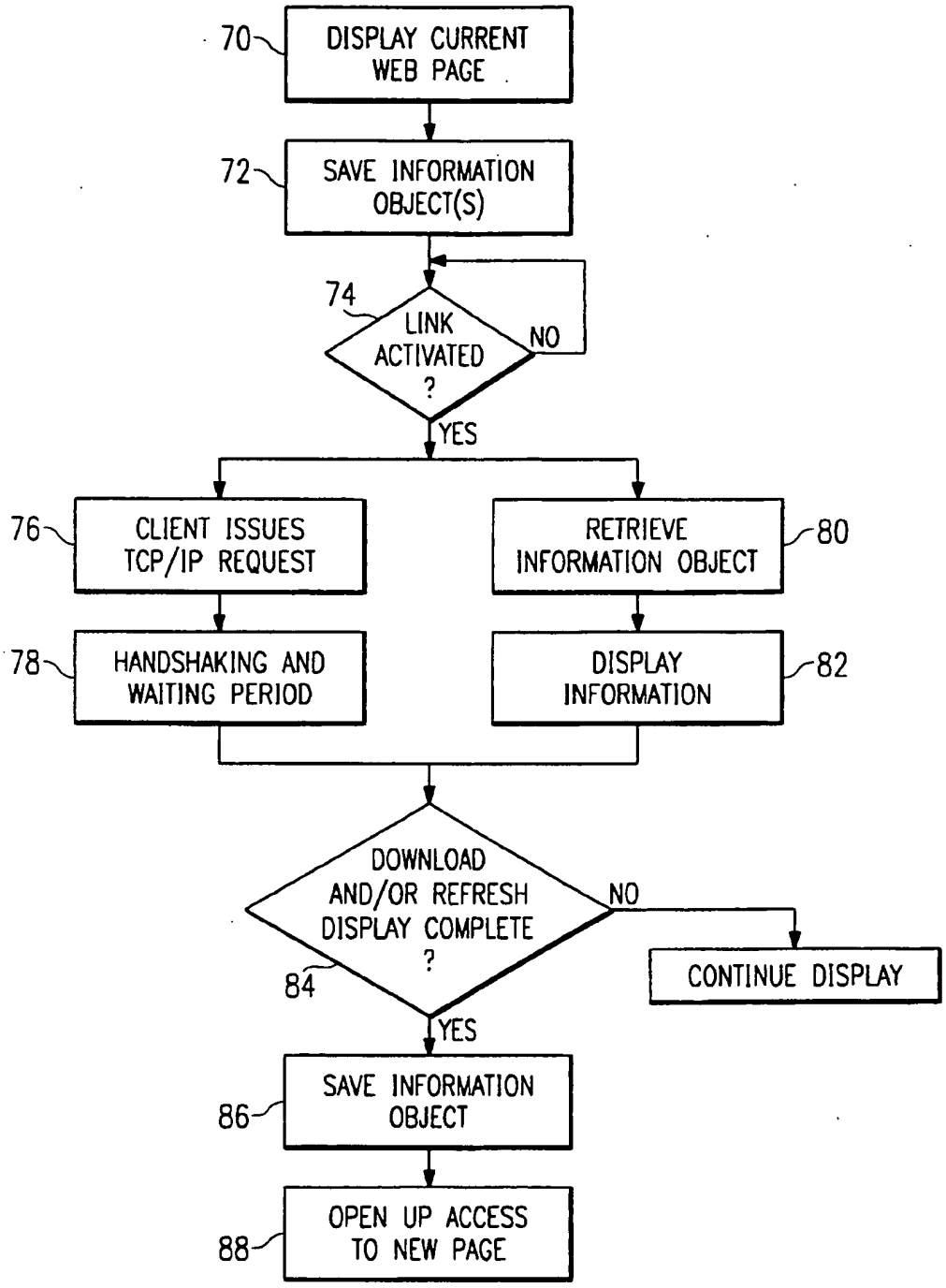
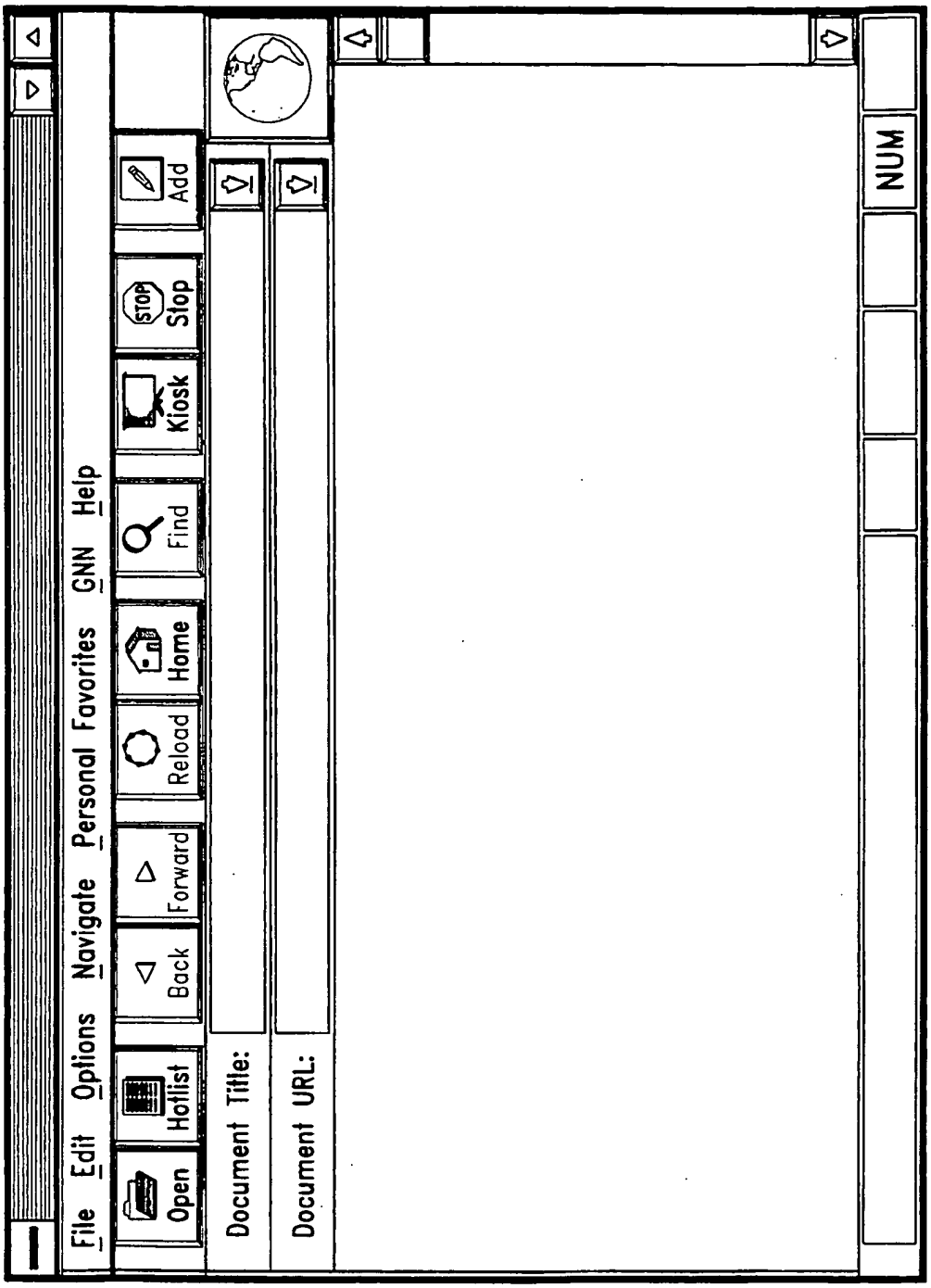

70
DISPLAY CURRENT WEB PAGE
72
SAVE INFORMATION OBJECT(S)
74
LINK ACTIVATED ?
NO
YES
76
CLIENT ISSUES TCP/IP REQUEST
78
HANDSHAKING AND WAITING PERIOD
80
RETRIEVE INFORMATION OBJECT
82
DISPLAY INFORMATION
DOWNLOAD AND/OR REFRESH DISPLAY COMPLETE ?
84
NO
CONTINUE DISPLAY
YES
86
SAVE INFORMATION OBJECT
88
OPEN UP ACCESS TO NEW PAGE

FIG. 3

FIG. 4


File Edit Options Navigate Personal Favorites GNN Help
Open
Hotlist
Back
Forward
Reload
Home
Find
Kiosk
Stop
Add
Document Title:
Document URL:
NUM

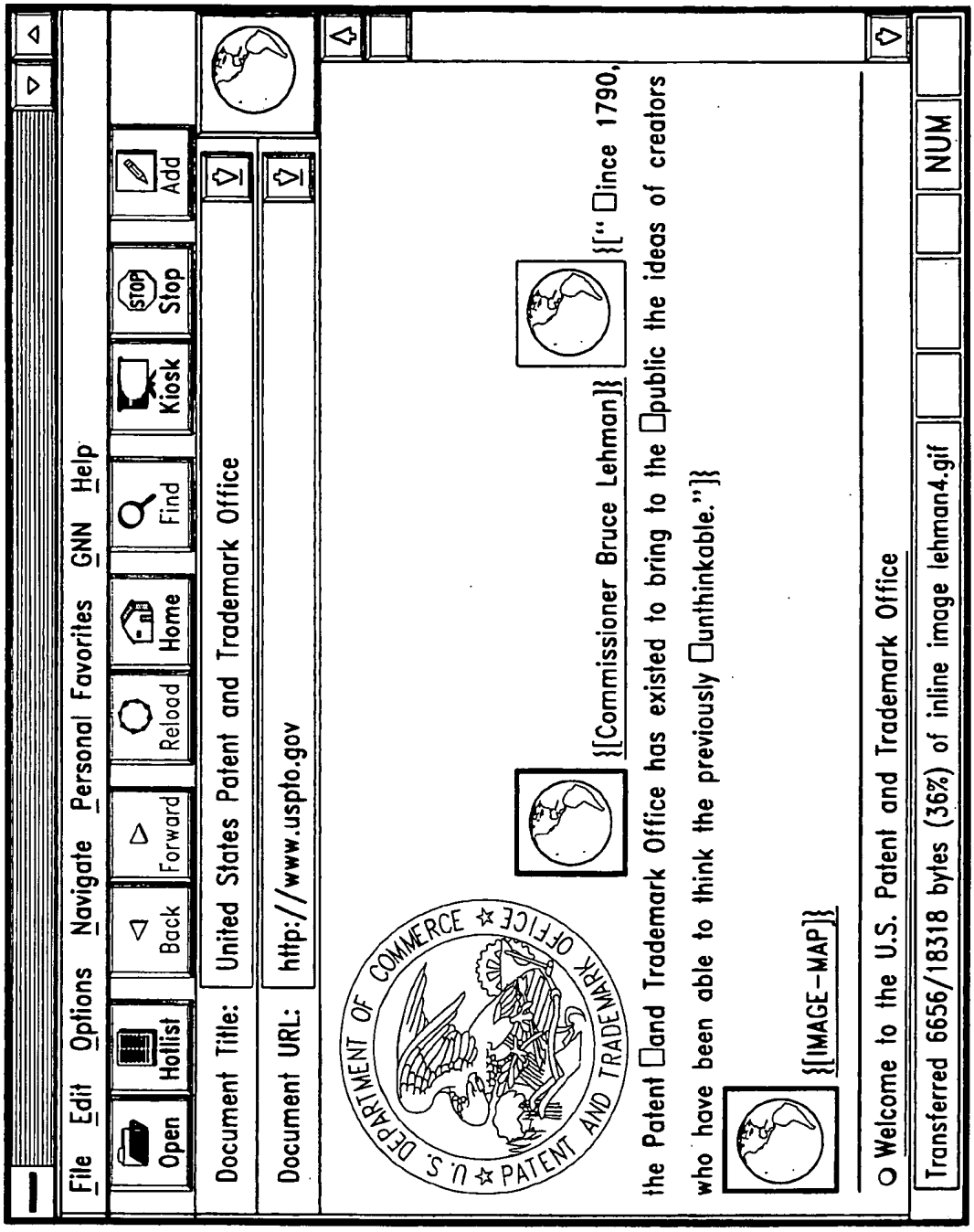

File Edit Options Navigate Personal Favorites GNN Help
Open
Hotlist
Back
Forward
Reload
Home
Find
Kiosk
Stop
Add
Document Title: United States Patent and Trademark Office
Document URL: http://www.uspto.gov
{[Commissioner Bruce Lehman]}
{[" Since 1790, the Patent and Trademark Office has existed to bring to the public the ideas of creators who have been able to think the previously unthinkable."]}
{[IMAGE-MAP]}
Welcome to the U.S. Patent and Trademark Office
Transferred 6656/18318 bytes (36%) of inline image lehman4.gif
NUM

FIG. 5

```
<HEAD>
<TITLE>United States Patent and Trademark Office</TITLE>
<LINK REV="OWNER" HREF="www@uspto.gov">
</HEAD>
<BODY>
<center>
<img width=172 height=164 src="ptog2.gif"
alt="[PTO Seal]">
<a href="http://www.uspto.gov/combio.html">
<img width=119 height=164 src="http://www.uspto.gov/lehman4.gif"
alt="[Commissioner Bruce Lehman]"></a>

<img width=231 height=139 src="http://www.uspto.gov/lequote.gif"
alt="[" ince 1790, the Patent
and Trademark Office has existed to bring to the
public the ideas of creators who have been able to think the previously
unthinkable."]">
<p>
<table border=5><tr><td
align=center><a href=http:www.uspto.gov/cgi-bin/imagemap/usptomap><img
border=0 width=569 height=240 src="usptomap.gif"
alt="[IMAGE-MAP]" ISMAP></a></td></tr></table>
</center>
<!--comment-->
<p><hr>
<ul>
<a href="http://www.uspto.gov/web/menu/menu1.html">
<li><b>Welcome to the U.S. Patent and Trademark Office
</b></a>
<!--comment-->
```

FIG. 6

```
<HEAD>
<TITLE>United States Patent and Trademark Office</TITLE>
<LINK REV="OWNER" HREF="www@uspto.gov">
</HEAD>
<BODY>
<center>
<img width=172 height=164 src="ptog2.gif"
alt="[PTO Seal]">
<a href="http://www.uspto.gov/combio.html">
<img width=119 height=164 src="http://www.uspto.gov/lehman4.gif"
alt="[Commissioner Bruce Lehman]"></a>

<img width=231 height=139 src="http://www.uspto.gov/lequote.gif"
alt="[" ince 1790, the Patent
and Trademark Office has existed to bring to the
public the ideas of creators who have been able to think the previously
unthinkable."]">
<p>
<table border=5><tr><td
align=center><a href=http:www.uspto.gov/cgi-bin/imagemap/usptomap><img
border=0 width=569 height=240 src="usptomap.gif"
alt="[IMAGE-MAP]" ISMAP></a></td></tr></table>
</center>
<!--comment-->
<p><hr>
<ul>
<a href="http://www.uspto.gov/web/menu/menu1.html">
<li><b>Welcome to the U.S. Patent and Trademark Office
</b></a>
<!--The PTO Welcomes You-->
```

FIG. 7

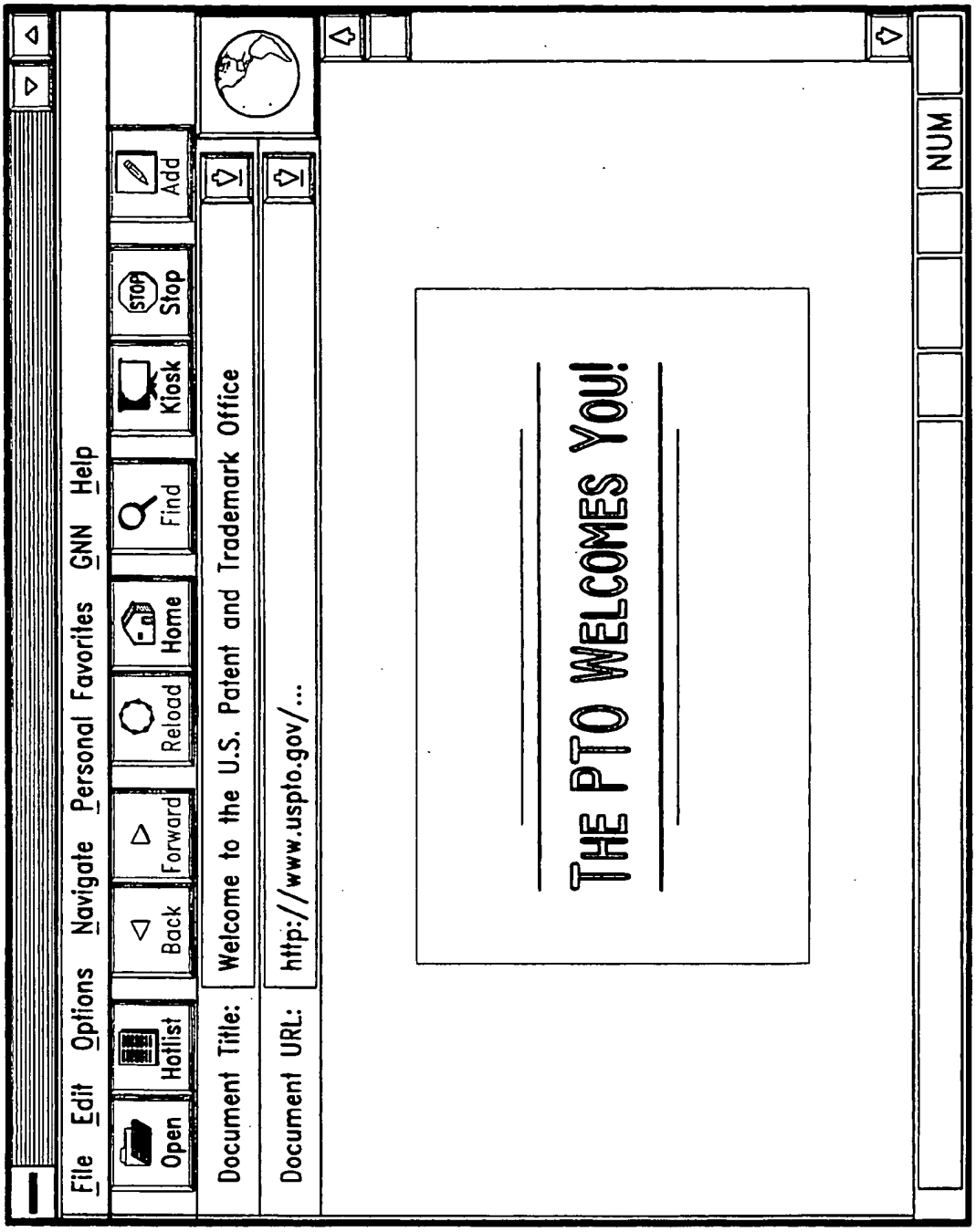

File Edit Options Navigate Personal Favorites GNN Help
Open
Hotlist
Back
Forward
Reload
Home
Find
Kiosk
Stop
Add
Document Title: Welcome to the U.S. Patent and Trademark Office
Document URL: http://www.uspto.gov/...
THE PTO WELCOMES YOU!
NUM

FIG. 8

WEB BROWSER WITH DYNAMIC DISPLAY OF INFORMATION OBJECTS DURING LINKING

TECHNICAL FIELD

This invention relates generally to computer networks and more particularly to methods for enhancing the operation of a client browser operating in a multi-server computer environment.

BACKGROUND OF THE INVENTION

The worldwide network of computers commonly known as the "Internet" has seen explosive growth in the last several years. Mainly, this growth has been fueled by the introduction and widespread use of so-called "web" browsers, which allow for simple graphical user interface (GUI)-based access to network servers, which support documents formatted as so-called "web pages". The "World Wide Web" (WWW) is that collection of servers of the Internet that utilize the Hypertext Transfer Protocol (HTTP). HTTP is a known application protocol that provides users access to files (which can be in different formats such as text, graphics, images, sound, video, etc.) using a standard page description language known as Hypertext Markup Language (HTML). HTML provides basic document formatting and allows the developer to specify "links" to other servers and files. Use of an HTML-compliant client browser involves specification of a link via a Uniform Resource Locator or "URL". Upon such specification, the client makes a tcp/ip request to the server identified in the link and receives a "web page" (namely, a document formatted according to HTML) in return.

There is a finite time period between the time the user initiates the link and the return of the web page. Even when the web page is returned quickly, there is an additional time period during which formatting information must be processed for display on the display interface. For example, most web browsers display in-line images (namely images next to text) using an X bit map (XBM) or .gif format. Each image takes time to process and slows downs the initial display of the document. The user typically "sees" an essentially unrecognizable "image" on the display screen which only gradually comes into focus. It is only after the entire image is downloaded from the server and then processed by the browser that the user can fully access the web page itself. This "waiting" period is even longer when the client machine has a relatively slow modem, and often the user will have to wait many seconds before being able to see the in-line image and/or begin using the web page. This problem will be exacerbated when the next generation browser technology (such as Netscape Navigator 2.0) becomes more widely implemented because such browsers are being designed to handle much more complex download formats (for more interactive, dynamic displays).

BRIEF SUMMARY OF THE INVENTION

It is thus a primary object of the invention to enhance the operation of a web browser by causing the display of some useful information to the user during the period of user "downtime" that otherwise occurs between linking and downloading of a hypertext document identified by the link. Such information may include, without limitation, advertisements, messages, fill-in forms, notices from a service provider, notices from another Internet service (such as receipt of an e-mail message), or some third party notice.

It is another more particular object of the invention to use an Hypertext Markup Language comment (e.g., via an HTML comment tag) in a web page to store an information object related to a link and then formatting and displaying such information when the link is activated.

It is still another object of the invention to embed an information object within an existing web page so that the object is masked until a link to another web page is activated. Upon activation, the object is displayed to the user effectively as a "mini" web page while the browser calls the link and awaits for a reply and download.

For example, in one particular embodiment, the information object includes copyright management information for a hypertext document associated with a link in a currently-displayed page. Such information may include the name or other identifying information of a copyright owner, terms and conditions for uses of the work within the hypertext document, and such other information as may be prescribed or desired. When the user "hits" the link in the current page, the copyright management information (which is already present in the browser) is displayed as the new document is being accessed and downloaded. The copyright management information, for example, may inform the user of the terms and conditions of how the copyrighted content being downloaded can then be reused. The "time" period normally associated with the download is thus productive for both the user (since he or she no longer has to sit and wait for the display) as well as to the content provider.

According to the preferred embodiment, there is described a method of browsing the Worldwide Web of the Internet using an HTML-compliant client supporting a graphical user interface and a browser. The method begins as a web page is being displayed on the graphical user interface, the web page having at least one link to a hypertext document preferably located at a remote server. In response to the user clicking on the link, the link is activated by the browser to thereby request downloading of the hypertext document from the remote server to the graphical user interface of the client. While the client waits for a reply and/or as the hypertext document is being downloaded, the browser displays one or more different types of informational messages to the user. Such messages include, without limitation, advertisements, notices, messages, fill-in forms, copyright information and the like. Preferably, the message information is in some way related to the hypertext document being accessed and downloaded, as in the case of copyright management information perhaps warning the user that the material being downloaded is subject to certain use restrictions of the copyright owner. Where the displayed information is related to the link, it is desirable that such information be embedded within the web page from which the link is launched. The information is preferably "hidden" within the web page using a hypertext markup comment tag.

The invention is preferably implemented in a computer having a processor, an operating system, a graphical user interface and a HTTP-compliant browser. In such case, the novel and advantageous features of the invention are achieved using a first means, responsive to activation of a link from a web page, for retrieving an information object masked within the web page, and a second means for displaying information from the information object on the graphical user interface as the browser establishes the link. Preferably, the information object is masked by an HTML comment tag, which may include other HTML tags nested therein to format the information in the object. This enables the support of complex "mini" web pages that are displayed and accessible to the viewer during otherwise nonproductive

periods when the browser is busy processing links to other documents or web sites.

The foregoing has outlined some of the more pertinent objects of the present invention. These objects should be construed to be merely illustrative of some of the more prominent features and applications of the invention. Many other beneficial results can be attained by applying the disclosed invention in a different manner or modifying the invention as will be described. Accordingly, other objects and a fuller understanding of the invention may be had by referring to the following Detailed Description of the preferred embodiment.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention and the advantages thereof, reference should be made to the following Detailed Description taken in connection with the accompanying drawings in which:

FIG. **1** illustrates a computer network in which the present invention is implemented;

FIG. **2** illustrates a client computer supporting an HTML-compliant Worldwide Web browser;

FIG. **3** is a flowchart diagram of a preferred method of the present invention for dynamic display of an information object during linking;

FIG. **4** is a representative graphical user interface illustrating browser navigation tools;

FIG. **5** is a representative web page illustrating a hypertext link;

FIG. **6** is a view of the HTML source code for the web page of FIG. **5**;

FIG. **7** is an example of a modified version of the HTML source code for the web page illustrated in FIG. **5**, showing an information object embedded therein through a comment tag; and

FIG. **8** is a representative screen display illustrating how the information object appears as a "mini" web page upon activation of the hypertext link in the web page of FIG. **5**.

DETAILED DESCRIPTION

As represented in FIG. **1**, the Internet is a known computer network based on the client-server model. Conceptually, the Internet comprises a large network of "servers" **10** which are accessible by "clients" **12**, typically users of personal computers, through some private Internet access provider **14** (such as Internet America) or an on-line service provider **16** (such as America On-Line, Prodigy, Compuserve, the Microsoft Network, and the like). Each of the clients may run a "browser," which is a known software tool used to access the servers via the access providers. A server **10** operates a so-called "web site" which supports files in the form of documents and pages. A network path to a server is identified by a so-called Uniform Resource Locator or URL having a known syntax for defining a network connection.

The "World Wide Web" (WWW) is that collection of servers of the Internet that utilize the Hypertext Transfer Protocol (HTTP). HTTP is a known application protocol that provides users access to files (which can be in different formats such as text, graphics, images, sound, video, etc.) using a standard page description language known as Hypertext Markup Language (HTML). HTML provides basic document formatting and allows the developer to specify "links" to other servers and files. Use of an HTML-compliant client browser involves specification of a link via the URL. Upon such specification, the client makes a tcp/ip request to the server identified in the link and receives a "web page" (namely, a document formatted according to HTML) in return.

FIG. **2** shows a block diagram of a representative "client" computer in which the present invention is implemented. The system unit **21** includes a system bus or plurality of system buses **31** to which various components are coupled and by which communication between the various components is accomplished. The microprocessor **32** is connected to the system bus **31** and is supported by read only memory (ROM) **33** and random access memory (RAM) **34** also connected to system bus **31**. The ROM **33** contains among other code the Basic Input-Output system (BIOS) which controls basic hardware operations such as the interaction and the disk drives and the keyboard. The RAM **34** is the main memory into which the operating system and application programs are loaded. The memory management chip **35** is connected to the system bus **31** and controls direct memory access operations including, passing data between the RAM **34** and hard disk drive **36** and floppy disk drive **37**. The CD ROM **42**, also coupled to the system bus **131**, is used to store a large amount of data, e.g., a multimedia program or large database.

Also connected to this system bus **31** are various I/O controllers: the keyboard controller **38**, the mouse controller **39**, the video controller **40**, and the audio controller **41**. The keyboard controller **38** provides the hardware interface for the keyboard **22**, the controller **39** provides the hardware interface for the mouse (or other point and click device) **23**, the video controller **40** is the hardware interface for the display **24**, and the audio controller **41** is the hardware interface for the multimedia speakers **25a** and **25b**. A modem **50** enables communication over a network **56** to other computers over the computer network.

The operating system **60** of the computer may be DOS, WINDOWS 3.x, WINDOWS '95, OS/2, AIX, or any other known and available operating system, and each computer is sometimes referred to as a machine. RAM **34** also supports a number of Internet access tools including, for example, the HTTP-compliant web browser **62**. Known browser software includes Netscape, Netscape Navigator **2.0**, Mosaic, and the like. The present invention is designed to operate within any of these known or developing web browsers, which are preferably modified as described herein to achieve the dynamic display of information objects during web site linking activities. RAM **34** may also support other Internet services including simple mail transfer protocol (SMTP) or e-mail, file transfer protocol (FTP), network news transfer protocol (NNTP) or "Usenet", and remote terminal access (Telnet).

HyperText Markup Language uses so-called "tags," denoted by the <> symbols, with the actual tag between the brackets. Most tags have a beginning (<tag>) and an ending section, with the end shown by the slash symbol (</tag>). There are numerous link tags in HTML to enable the viewer of the document to jump to another place in the same document, to jump to the top of another document, to jump to a specific place in another document, or to create and jump to a remote link (via a new URL) to another server. Links are typically displayed on a web page in color or with an underscore. In response to the user pointing and clicking on the link, the link is said to be "activated" to begin the download of the linked document or text. For more details on HTML, the reader is directed to the *HTML Reference Manual*, published by Sandia National Laboratories, and available on the Internet at "http://www.sandia.gov/sci_

compute/html.ref.html" or the *HTML Quick Reference*, published by the University of Kansas, and available on the Internet at "http://kuhttp.cc.u.kans.edu/lynx_help/HTML_quick.html". Each of these publications are incorporated herein by reference.

A known HTML tag is a "comment," which typically allows a web page developer to include text that is to be ignored by the browser. The syntax for a "comment" tag is denoted <!--text-->. HTML is an evolving language. Recent standards for new versions of this language propose to add SGML comment syntax to HTML elements. This proposal would begin a comment with a double dash encountered inside any HTML element (but no inside quotes), and treat every thing as comments (including any " "," ", or quote character) until the next occurring double dash. Such syntax allows HTML elements within a comment.

According to the present invention, an information "object" is preferably placed within a comment tag of a web page and thus is "ignored" by the browser in the formatting of the document then being displayed. This information object, however, is also saved to a separate file or cache within the client. A particular web page may have multiple information objects, with one or more objects associated with one or more links in the documents. Thus, for example, if the document has two links, one information object is associated with the first link and a second information object is associated with a second link, and so on. Or, multiple information objects may be associated with a single link. Or, the information object(s) may have no direct relation to the content of any link in the document. While in the preferred embodiment an HTML "comment" tag is used to mask the information object, those skilled in the art will recognize that other HTML commands and tags may be used for this purpose as well including, for example, tag dedicated to masking an information object within the currently-displayed page. For example, an information object may be hidden within a clickable image identified with an ismap tag. Also, an information object may be formatted as a "mini" web page by nesting HTML elements within a particular HTML comment tag.

As noted above, a web browser **62** running on the client uses a TCP/IP connection to pass a request to a web server running a HTTP "service" (under the WINDOWS operating system) or "daemon" (under the UNIX operating system). The HTTP service then responds to the request, typically by sending a "web page" formatted in the Hypertext Markup Language, or HTML, to the browser. The browser then displays the web page using local resources (e.g., fonts and colors).

A preferred operation of the inventive method is illustrated in the flowchart of FIG. **3**. The method begins at step **70** as a current web page is being displayed on the graphical user interface of the computer. It is assumed that this web page has embedded therein one or more comment tags, each of which (or perhaps several of which in combination) define an information object. Generally, although not required, each information object will be provided for one or more links in the web page being displayed. However, because the information object is embedded within a comment tag, it is hidden or "masked" and thus is ignored by the display routines of the browser. In step **72**, the method saves or stores the information object in memory or some dedicated portion of the RAM (e.g., a cache) so that it may be easily and quickly obtained. At step **74**, a test is made to determine whether a link associated with the information object has been activated. If so, the method continues at step **76** and issues a tcp/ip request to the network (assuming the link was to a URL). Step **78** represents the handshaking period during which the client waits for the appropriate response from the server. During this period, the client retrieves the information object (at step **80**) and outputs the information (in step **82**) to the user on the display. Steps **80** and **82** are shown in parallel to the handshaking and wait step **78** to emphasize the inventive concept of displaying useful information to the viewer during the link process. At step **84**, a test is then performed to determine whether the download and refresh of the display is complete. If so, the routine saves the information object at step **86** and opens up access to the hypertext document at step **88**.

FIG. **4** shows the browser navigation tool prior to download of the U.S. Patent and Trademark Office page (available at http://www.uspto.gov). FIG. **5** shows the web page as it exists on the display. This web page has various links including "Welcome to the United States Patent and Trademark Office." FIG. **6** shows the HTML source code used to generate the web page of FIG. **5**, and FIG. **7** shows this source code modified to include an information object **75** within a comment tag. This object displays the message "The PTO Welcomes You" when the "Welcome to the United States Patent and Trademark Office" link is activated. FIG. **8** shows the effect of this information object when the routine of FIG. **3** is carried out.

Although the invention has been described in terms of a preferred embodiment, those skilled in the art will recognize that various modifications of the invention can be practiced within the spirit and scope of the appended claims. Thus, for example, the information supplied to the user during the period between link activation and downloading of the hypertext document need not be merely a visual output. It is also envisioned that some or all parts of a particular message be conveyed to the user aurally (via a multimedia speaker set, for example) as well as on the display screen. The message itself may be retained on the screen as an inline image or other text along with the downloaded hypertext document, and the browser includes appropriate means to queue the message to print and/or to save the message or allow the user to compose a response to the message. One such technique for responding to the message uses the HTML "fill-in" form tags. The browser may be suitably programmed to queue the mini web page for background printing whenever the link is activated.

Moreover, although in the preferred embodiment it is envisioned that the hypertext document (associated with the web page link) is located on a remote server, this is not a limitation of the invention. The display of informational messages may be effected whenever a link is activated, regardless of the location of the target document. Also, while the preferred embodiment has been described in the context of an Internet browser, the techniques of the invention apply whether or not the user accesses the Worldwide Web via a direct Internet connection (namely using an Internet access provider) or indirectly through some on-line service provider (such as America On-Line, Prodigy, Compuserve, the Microsoft Network, or the like). Thus the "computer network" in which the invention is implemented should be broadly construed to include any client-server model from which a client can link to a "remote" document (even if that document is available on the same machine or system).

It should also be appreciated that while in the preferred embodiment the information object is formatted and displayed upon activation of a link in a web page being currently displayed, this is not a limitation of the invention either. The information object need not be embedded within an existing web page, but rather may be embedded within

the home page of the browser or supported elsewhere within the client itself. Thus, the information object may be displayed whenever a call to a web page is made, such as when a search to a particular URL is initiated, or when a previously-stored URL is launched (e.g., from a "View Bookmark" pulldown menu). Moreover, the client may store random information objects in the form of information advertisements (which in turn may include .gif files to produce images) so that the browser may call any such information at random. The browser may even be programmed to select which of the plurality of information objects to display based on a comparison of the type of web pages accessed by the user. Thus, for example, if the user accesses web pages relating to a particular service, the browser may be programmed to identify this access history and select predetermined information objects that will be of interest to the user (given that history).

The information objects may themselves be HTML "fill-in" forms that are retained on the display screen and may be filled in with information that the browser can then deliver back to some third party service provider. This enables the information objects to be used as mini survey forms for interactive, on-line surveys and the like, which would be especially advantageous for web site providers or other third parties.

As noted above, the information object may be automatically or selectively queued to the client printer upon display. This would enable the viewer to generate merchandise coupons and the like related to the web page being accessed. Thus the web site provider could offer the viewer some added incentive for accessing its web page by causing the printing of a redeemable coupon or other information token (e.g., a discount card, a receipt, etc.). All of these actions are initiated during the otherwise downtime between web page access and download, thereby significantly increasing the value of the on-line informational content provided to the user.

As used herein, the "information object" or "information" output to the viewer during the link process should be broadly construed to cover any and all forms of messages, notices, text, graphics, sound, video, tables, diagrams, applets and other content, and combinations of any of the above.

One of the preferred implementations of the "browser" of the invention is as a set of instructions in a code module resident in the random access memory of the user's personal computer. Until required by the computer, the set of instructions may be stored in another computer memory, for example, in a hard disk drive, or in a removable memory such as an optical disk (for eventual use in a CD ROM) or floppy disk (for eventual use in a floppy disk drive). In addition, although the various methods described are conveniently implemented in a general purpose computer selectively activated or reconfigured by software, one of ordinary skill in the art would also recognize that such methods may be carried out in hardware, in firmware, or in more specialized apparatus constructed to perform the required method steps.

Finally, the present invention is designed to be implemented with conventional HTML and enhancements thereto (including HTML 2.0, HTML 3.0, HTML with third party-supplied extensions such as NHTML, and the like), by an HTML-compliant browser, such as Netscape, Netscape Navigator 2.0, Mosaic, MSN, as such existing or developed programs are modified to include the functionality of the invention described above. Netscape Navigator 2.0 has in-line support for platform-independent application objects (e.g., applets written in JavaScript, from Sun Microsystems). An applet resides on the server associated with a web page and is downloaded to the client browser after a link is established to the web page. The browser includes an engine for executing the downloaded applets. With this type of browser, the invention caches or otherwise stores a downloaded applet and later uses it, preferably when a new, related link is established. Thus, an "information object" according to the invention may include an applet which, for example, may generate an animated figure or icon, some aural output, a scrolling display, or a combination thereof. One of ordinary skill, however, will recognize that the inventive features of the invention, including the masking of "mini" hypertext documents within a web page and display of such documents upon link activation, may be applied to other Internet services as well as to HTTP compliant browsers. Thus, the invention would be useful to provide information to a user during an FTP access, an on-line chat, a posting to a bulletin board, or even during the sending and retrieval of e-mail. All such variations are considered within the scope of the invention.

Having thus described my invention, what I claim as new and desire to secure by Letters Patent is set forth in the following claims.

What is claimed is:

1. A method of browsing in a computer network having at least one client connectable to one or more servers, the client having an interface for displaying a first hypertext document with at least one link to a second hypertext document located at a server, comprising the steps of;

contacting the server in response to activation of the link to initiate downloading of the second hypertext document from the server to the client;

in response to activation of the link, retrieving an information object that has been stored in the client prior to the activation of the link; and

displaying the retrieved information object on the interface during at least a portion of a time period between the activation of the link and completion of the downloading of the second hypertext document from the server to the client to provide information to a user during a process of linking from the first hypertext document to the second hypertext document;

wherein the information object is stored within the first hypertext document and is not displayed on the interface until after activation of the link that initiates downloading of the second hypertext document from the server to the client.

2. The method of browsing of claim 1 wherein the information object is stored within the first hypertext document using a markup tag that normally specifies a comment within a hypertext document.

3. The method of browsing of claim 2 wherein the markup tag includes nested HTML tags to format the information object within the markup tag.

4. A method of browsing in a computer network having at least one client connectable to one or more servers, the client having an interface for displaying a first hypertext document with at least one link to a second hypertext document located at a server, comprising the steps of:

contacting the server in response to activation of the link to initiate downloading of the second hypertext document from the server to the client;

in response to activation of the link, retrieving an information object that has been stored in the client prior to the activation of the link; and

displaying the retrieved information object on the interface during at least a portion of a time period between the activation of the link and completion of the downloading of the second hypertext document from the server to the client to provide information to a user during a process of linking from the first hypertext document to the second hypertext document;

wherein the information object includes copyright management information for the second hypertext document.

5. The method of browsing of claim 4 wherein the copyright management information includes terms and conditions for use of a copyrighted work being downloaded from the server to the client.

6. A method of browsing in a computer network having at least one client connectable to one or more servers, the client having an interface for displaying a first hypertext document with at least one link to a second hypertext document located at a server, comprising the steps of:

contacting the server in response to activation of the link to initiate downloading of the second hypertext document from the server to the client;

in response to activation of the link, retrieving an information object that has been stored in the client prior to the activation of the link; and

displaying the retrieved information object on the interface during at least a portion of a time period between the activation of the link and completion of the downloading of the second hypertext document from the server to the client to provide information to a user during a process of linking from the first hypertext document to the second hypertext document;

wherein the information object includes an advertisement.

7. The method of browsing of claim 6 wherein the advertisement advertises goods or services described in the second hypertext document.

8. The method of browsing of claim 7 further including the step of queueing the information object to a printer associated with the client to thereby print a coupon for the goods or services.

9. A method of browsing the World Wide Web of the Internet using a client machine supporting a browser, comprising the steps of:

storing an information object;

activating a link from a first hypertext document to initiate downloading of a second hypertext document;

retrieving the stored information object in response to activating the link; and

displaying the information object during at least a portion of a time period between activation of the link and completion of the downloading of the second hypertext document to provide information to a user of the client machine as the browser links from the first hypertext document to the second hypertext document;

wherein the information object is stored within the first hypertext document and is not displayed until after activating the link that initiates downloading of the second hypertext document.

10. The method of browsing of claim 9 further including the step of queueing the imformation object to a printer.

11. The method of browsing of claim 10 wherein the information object is an advertisement.

12. In a computer having a processor, an operating system, a graphical user interface and a browser, the improvement comprising:

means, responsive to activation of a link from a web page that initiates downloading of a linked hypertext document, for retrieving information masked within the web page; and

means for outputting the information during at least a portion of a time period between activation of the link and completion of the downloading of the linked hypertext document to provide information a user of the computer as the browser links from the web page to the linked hypertext document.

13. In the computer system as described in claim 12 wherein the information is outputted on the graphical user interface.

14. In the computer system as described in claim 12 wherein the information is masked by an HTML comment tag.

15. In the computer system as described in claim 12 wherein the HTML comment tag includes HTML tags nested therein to format the information.

16. In a computer having a browser for retrieving hypertext objects from servers in a computer network, the browser including an engine for executing applets, the improvement comprising:

means, responsive to activation of a link from a hypertext object that initiates downloading of a linked hypertext object, for retrieving an applet downloaded to the computer prior to activation of the link; and

means for outputting the applet during at least a portion of a time period between activation of the link and completion of the downloading of the linked hypertext object to thereby provide information to a user of the computer as the browser links from the hypertext object to the linked hypertext object.

17. In a computer having a browser for retrieving hypertext objects from servers in a computer network, the improvement comprising:

means, responsive to activation of a link from a hypertext object that initiates downloading of a linked hypertext object, for retrieving an information object downloaded to the computer prior to activation of the link; and

means for outputting the information object during at least a portion of a time period between activation of the link and completion of the downloading of the linked hypertext object to thereby provide information to a user of the computer as the browser links from the hypertext object to the linked hypertext object:

wherein the information object is an advertisement.

18. The computer as described in claim 17 wherein the advertisement is selected based on the link.

19. A method of browsing in a computer network having a plurality of servers that support hypertext objects, comprising the steps of:

storing a plurality of information objects at a client machine having a browser, wherein the information objects are downloaded to the client machine from the computer network;

in response to activation of a link that initiates downloading of a hypertext object, selectively displaying one of the stored information objects during at least a portion of a time period between activation of the link and completion of the downloading of the hypertext object to provide information to a user of the client machine during a process of linking to the hypertext object;

wherein the information objects are advertisements for goods or services.

* * * * *